A Model Driven Laboratory Information Management System

Hao Li

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington

2011

Program Authorized to Offer Degree:
Medical Education and Biomedical Health Informatics

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

University of Washington
Graduate School

This is to certify that I have examined this copy of a doctoral dissertation by

Hao Li

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examination committee have been made.

Chair of the Supervisory Committee:

James F. Brinkley

Reading Committee:

James F. Brinkley

Ira J Kalet
Ira J. Kalet

Linda Shapiro
Linda G. Shapiro

Date 2/28/2011

In presenting this dissertation in partial fulfillment of the requirements for the doctoral degree at the University of Washington, I agree that the Library shall make its copies freely available for inspection. I further agree that extensive copying of the dissertation is allowable only for scholarly purposes, consistent with “fair use” as prescribed in the U.S. Copyright Law. Requests for copying or reproduction of this dissertation may be referred to ProQuest Information and Learning, 300 North Zeeb Road, Ann Arbor, MI 48106-1346, 1-800-521-0600, to whom the author has granted “the right to reproduce and sell (a) copies of the manuscript in microform and/or (b) printed copies of the manuscript made from microform.”

Signature____________________

Date 2/28/2011

University of Washington

Abstract

A Model Driven Laboratory Information Management System

Hao Li

Chair of the Supervisory Committee:
Professor James F. Brinkley
Department of Medical Education and Biomedical Health Informatics

Biomedical research scientists need more robust tools than spreadsheets to manage their data. However, no suitable laboratory information management systems (LIMS) are readily available; they are either too costly to build or too complex to adapt. This thesis presents the architecture, design, implementation, and a prototype of a model driven LIMS, called Seedpod. Scientists, with the help of biomedical informaticists, develop a knowledge model of their data and data management needs in a knowledge management tool called Protégé. Seedpod then automatically produces a relational database from the model, and dynamically generates a web-based graphical user interface. Seedpod can be used for multiple scientific research domains since only its knowledge model contains domain-specific content. It decreases development time and cost, thereby allowing scientists to focus on producing and collecting data.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

This thesis would not have been possible without my advisor, Dr. James Brinkley, for his guidance, patience, trust, and wisdom through this near-decade journey. I have been incredibly fortunate and honored to have Dr. Ira Kalet and Dr. Linda Shapiro on my committee. They showered me with encouragement and inspiration that I needed both professionally and personally. I am indebted to Dr. Peter Mork and Dr. John Gennari for their collaboration in the model transformation component of the project, which became an impetus for my thesis project. Special thanks to Dr Micki Kedzierski from the Pharmacy School who shared with me her courage and provided unconditional support. My supportive colleagues at MITRE have inspired me to finish this dissertation. I cannot appreciate my dear friends enough for keeping me sane, balanced, happy, and optimistic in this endeavor. Last but not least, I owe my deepest gratitude to my family for their continuing love and support, especially my parents Dr. Lian Liang, Dai-zong Li, and Patrick Pang.

1. INTRODUCTION

1.1. Motivation

In the age of exponential growth of data, scientific research has evolved from being hypothesis-driven to being data-driven. Scientific discoveries rely on the ability to collect, manage, analyze, and make sense of large, rich, and complex multimedia datasets (Kell & Oliver, 2004; Drexler, 2008; Larson, 2008; Gray, Liu, Nieto-Santisteban, Szalay, DeWitt, & Heber, 2005). Using an Excel spreadsheet to manage experiment data is cumbersome, error prone and time consuming, and furthermore, it is limited to tabular data (Jakobovits, Rosse, & Brinkley, 2002; Fong & Brinkley, 2006). Advanced laboratory information management systems (LIMS) combining sophisticated computer tools, such as web applications and relational databases, are ubiquitous (Lacroix & Critchlow, 2003; Paszko & Turner, 2002.; Kotter, 2001; Gardner & Shepherd, 2004). However, development of such systems is costly in time and effort, so scientists rely on biomedical informaticists or computer engineers to develop them.

Frequent changes to experimental protocols in scientific research further complicate the data management problem (Jakobovits, Rosse, & Brinkley, 2002). With current approaches to developing and managing data management systems, informaticists[1]

[1] "Informaticist" will be used in place of "biomedical informaticist" for the rest of the thesis.

cannot make changes to the systems quickly enough to match the rate at which the experiments change. As a result data management is interrupted or slowed to a halt.

Various LIMS research and development efforts focus on cutting development cost and time, but most lack the ability to change for two reasons. The first is that LIMS are developed with a tight coupling of data collection with data analysis. Data analysis adds restrictions on data formats and storage methods for data collection. These two activities may and should occur independently, so that more data can be quickly collected without delay (Swenson, 2005). The second reason is that the changing components, most frequently the data model, of LIMS are fragmented and embedded in various components of the system (Schmidt, 2006). Changing an experimental protocol usually means making changes to the data model. This often requires the system database, application code, and logic to be changed throughout.

1.2. Problem Statement

The goal of this thesis is to develop a general and cost-effective LIMS development methodology that encapsulates the changing components of the LIMS in a descriptive model and automatically generates the LIMS data storage and graphical user interface based on the model.

1.3. Approach

This thesis is based on an existing software system technique called a model-driven approach (MDA) (MDA, 2010). The model is a descriptive representation of a LIMS including data elements, application logic, and presentation attributes. The application engine automatically translates the model to a relational database model. The web application server translates the model and dynamically generates a web-based user interface for users to manage the data in the relational database. The advantage of this approach is its cost saving. The unique element of this approach is the separation of domain-dependent knowledge model from domain-independent programming code.

1.4. Thesis Outline

The thesis is laid out as follows. Chapter 2 provides a detailed description of the scientific data management problem with example challenges from both scientists and informaticists and a list of requirements for the system. Chapter 3 evaluates existing LIMS development approaches and makes an argument that MDA is the superior approach, but a better MDA approach than existing methods is needed. Chapter 4 details the design, implementation, and result of a model-driven LIMS prototype called Seedpod. Seedpod contains three components: a LIMS model developed in a knowledge management tool called Protégé (Stanford Center for Biomedical Informatics Research, 2010), a transformation engine, and a web application engine. A methodology for automatic transformation of the Protégé model to a relational model is defined in Chapter 5. Chapter

6 evaluates the system against LIMS requirements from Chapter 2. The thesis concludes with contributions and future work in Chapter 7.

2. SUPPORTING SCIENTIFIC DATA MANAGEMENT

The advent of scientific recording techniques has resulted in an explosion of scientific data. Most of the significant discoveries are made in small to mid-sized research laboratories. Data management is the foundation of scientific research and laboratory experiments. The state-of-the-art practice in many research labs is to use basic Excel sheets or Access on a personal computer (Anderson, et al., 2007). Managing large volume and multimedia data with these tools is no longer feasible. Increasingly, researchers need to collaborate with each other over geographic distances which require them to leverage Internet technology (Gardner & Shepherd, 2004; Jakobovits, Soderland, Taira, & Brinkley, 2000). Informaticists resort to a mongrel cocktail of infrastructure and available tools to create solutions that are difficult to maintain and change (Swenson, 2005). Thus, LIMS for data management remains a bottleneck to biomedical research.

Information management challenges can be categorized according to whether they deal with data management within a single scientific laboratory, data sharing among interdisciplinary labs, and knowledge sharing. The focus of this thesis project is data management within a single or small group of labs: to capture, organize, and allow access to the data. The challenges and issues of data management in a university research setting have been well studied (Anderson, et al., 2007). In sections 2.1 and 2.2, challenges of data management are described from the perspectives of researchers and informaticists. These challenges allude to the requirements for a new solution in Section 2.3.

2.1. Challenges of Scientific Data Management for Researchers

Small to mid-sized biomedical research labs are in need of more robust data management support beyond spreadsheets, but they have limited access to informatics support. This section describes the Ojemann Lab at the University of Washington Medical Center (UWMC), Department of Neurosurgery, to illustrate the main issues that biomedical research laboratories face. A second example is provided from the Stevens Lupus clinical research lab at Seattle Children's Hospital. Both of these examples are based on the author's observations. Whether it is clinical research or basic science research, many of the challenges faced by these small to mid-sized scale university research labs are representative. In addition to technical challenges, challenges that enable stakeholders to work together are discussed as well (Anderson, et al., 2007; Jakobovits, Soderland, Taira, & Brinkley, 2000).

2.1.1. Example #1: Single Unit Recording at the Ojemann Lab

The Ojemann Lab studies the relationship between language memory and functional organization of language related neurons in the temporal cortex of the human brain (Ojemann, Schoenfield-McNeill, & Corina, 2002). This is the only laboratory in the U.S. that records from a live human brain using an electrophysiological recording technique called single unit recording (SUR). Unlike other non-invasive recording techniques such as functional magnetic resonance imaging (fMRI), electroencephalography (EEG), and positron emission tomography (PET), SUR has the advantage of high spatial and temporal resolution for direct correlation between the stimuli and the observed activation.

Therefore, SUR experiments produce valuable data giving insights into the functional organization of the language cortex that other techniques do not.

Ojemann's SUR experiments take place during epileptic resection surgeries. Tungsten electrodes record extracellularly from the cortical areas of a human subject. During a surgery, the patient subject performs a sequence of language tasks, or trials, while the microelectrodes record simultaneously from the temporal lobe of the brain. The language tasks are preplanned using a psychology experiment design and operating system called E-Prime (Psychology Software Tools, Inc). Each task contains one or more stimuli items that may be presented in textual, auditory, or pictorial forms. The patient then responds by either identifying or remembering the stimuli according to instructions for each trial. If any language errors occur, the subject's responses are documented on a paper log sheet. Meanwhile, small electrical signals that mark the stimuli onset time and patient's response time are sent to another computer running software called Chart. Chart is a software program developed by ADInstrument (ADInstruments), which is commonly used by electrical physiologists to record from neurons. Chart records simultaneously from signals produced by multiple channels of the electrodes at high resolution.

Data organization and management take place after the experiments. The raw data are archived on CDs. The saved Chart files with signal recordings are filtered and processed using a MatLab program to remove artifacts and signal noise. Time series data are parsed for individual neurons recorded by multiple microelectrodes. They are then saved into new individual files. Time series files for stimulus onset and patient responses are also saved separately from the neuron responses. Each of the electrode time series files is processed by a MatLab spike sorting program. This program differentiates the neurons that an electrode records from by signal amplitudes (Cho, Corina, Brinkley, Ojemann, & Shapiro, 2005). Individual neuron time series are then saved. Finally, the neuronal time series are parsed by trial onset time series, and the frequency of each neuron's response to each trial is calculated. Through this process, some of the data are processed by PowerLab (ADInstruments) and some are processed by a data analyst who writes signal processing programs, which may perform better to meet the needs of the lab. Multiple visualizations of the neuronal signals, such as raster plots and neuronal response histograms by different time bin sizes, are generated to facilitate data processing. At the end of an experiment, several different kinds of data artifacts are generated and stored in files. Sometimes, multiple formats of the files are stored for various researchers that use different computing platforms such as Mac or PC. These artifacts are organized by experiment protocols and subjects. Each subject directory takes up to 1.4 GB on a remote hard drive, in addition to the CD archives, and multiple copies of the data are stored on local folders of different researchers. The files names are concatenated identifiers assigned at each process step to help researchers recognize them quickly. Other data collected or derived through the experiment such as patient demographics, experiment notes, frequencies are organized in Excel spreadsheets.

Research staff at the Ojemann lab must be very meticulous about data management using Excel sheets and ad hoc methods. They must coordinate to work with each other with a complicated workflow. Data access is fragmented. Data are collected from different instruments and sources, and then stored in multiple media, such as spreadsheets, CD archives, remote file management hard drives, and paper lab notes. Different researchers in the lab use different platforms such as Windows versus Mac. Sometimes files need to be saved twice for the different platforms. Multimedia data metadata management is not available. This makes searching for files difficult if not impossible. Most of the data are stored on local hard drives, making data entry, remote access, data sharing, and version control between several people prone to error and non-feasible. Along with this type of ad hoc data management, disparate users may or may not conform to file naming conventions or other data entry standards, which lowers data quality, correctness and completeness. As is inherent to non-structured data storage, search and retrieving multimedia data is manual and time consuming. With increasing data size, the amount of manual work to clean up data for analysis becomes exponentially more cumbersome.

2.1.2. Example #2: Lupus Study at the Stevens Lab

Dr. Anne Stevens is a clinician and researcher at Seattle Children's Hospital. She studies maternal mitochondria genetic inheritance effects in Lupus. Akin to Dr. Ojemann's Lab, she has a staff of researchers working for her gathering data from various sources. Data are consolidated and organized into Microsoft Excel spreadsheets from Children's Hospital databases, interviews with the patients, and Fred Hutchinson Cancer Research Institute (FHCR). It is with FHCR that she shares her subject data. Data from Children's Hospital are collected from three different databases due to disparate data storage from different clinics and departments in the hospital at the time of the interview. Data quality control is challenging when multiple researchers need to access the same data from various locations. Managing multimedia data is not a huge problem. However, like Dr.

Ojemann's Lab, querying data from various spreadsheets for data analysis is cumbersome and time consuming.

2.1.3. Section Summary

This section describes the role of scientific users with LIMS. They plan, conduct and manage experiments. Their interest is in research, not data management or information technology. They are intimately familiar with the data structure and domain knowledge. They prefer to have control of the data (Gray, Liu, Nieto-Santisteban, Szalay, DeWitt, & Heber, 2005) . They are often willing to use any solutions that help them manage data even if the solutions are inefficient. They are limited in financial resources, technical staff, and time invested into LIMS development or maintenance. Their willingness to compromise with the use of cumbersome tools is justified by the control they gain by using more simple solutions (Lazar, 2000). This eventually becomes an issue between scientist users and LIMS developers when user involvement in design and implementation of the system is minimized.

From the two examples provided in this section, despite the differences in their research fields, there are common data management issues. These challenges are part technical and part organizational. The technical challenges involve scaling the solution to an expanding data set. The organizational challenges involve coordinating the research staff to better collaborate and share data management tasks.

2.2. Challenges of Data Management for Informaticists

Biomedical informaticists are another group of LIMS stakeholders that one must consider when deciding which LIMS to use. Different levels of technical skill sets come with different informaticists or IT professionals. The level of interaction between scientific users and informaticists determines how independent scientific users can be with the system. In this section, challenges faced by informaticists are described from development to maintenance from an example of Brinkley's Structural Informatics Group (SIG) at the UWMC (Structural Informatics Group). SIG developed and maintains LIMS for Ojemann's and Steven's labs as mentioned above.

2.2.1. Custom solution development

SIG has a small group of computer developers with special interests in scientific data management. The group has built a slew of LIMS for quite a few research laboratories. Once the systems have been built, SIG continues to maintain these systems over the years. LIMS development is costly and time consuming. The application developers must spend a significant amount of time up front to understand the domain science and information program. Then they design an information system to meet the users' needs. Once a system is architected, the developer designs a data model which is used for implementing a database schema. A web-based application allowing users to manage data through a web browser is preferred, because its development cost is low and it provides multi-user remote access. The data management system is highly customized for individual laboratories. Therefore, both the development and on-going maintenance is costly.

2.2.2. Application evolution

With increasing experience in developing and managing research LIMS, SIG developers observed development patterns. Based on these patterns, tools with some level of reusability were born. Reusable and customizable development modules can help to speed up the development process, therefore cutting down development cost. The tools are less domain-specific. However, these programming modules are too complicated for a scientific user to grasp and use, so SIG must be dedicated to ongoing maintenance activities.

Unlike a regular chemistry lab with a fairly routine and standardized protocol, scientific research demands frequent change to its experimental protocol. The changes may take place for as short as 3 months apart to a year. The data management system is also expected to change to meet new needs of new protocols. However, evolving an existing system is not a simple task. The database may need to change its schema and pre-existing data. The server application that dynamically generates the web front-end populated with data in the database may need to be modified. System evolution may be as costly as developing from the start in terms of manpower and time. Quickly rolling out new versions in a highly volatile changing environment is difficult given SIG's available resources.

2.2.3. Supporting multiple laboratories

Because SIG develops and maintain multiple laboratories' data management systems, it is in a unique position to reuse tools it builds for one laboratory in another. In fact, scientists from one study expressed the need for institution-wide technical support

(Anderson, et al., 2007). With added laboratories SIG would need to employ more engineers to develop and then continue to dedicate more hours for maintenance. If the projects at SIG grew without growing the number of engineers, the time for changes to a system to take place would become longer. This is not acceptable for scientific data management systems, which demands frequent changes.

2.2.4. Section summary

Informaticists develop databases and user interface tools for scientists to access and manage data. They study the domain science information problems and develop computer solutions to address the problems appropriately. They are also responsible for maintaining and evolving the applications when experimental protocols change over time. It is time consuming and costly to make changes to an existing data management system. The problem is compounded by supporting multiple customized systems.

2.3. System Requirements and Evaluation Plan

The challenges in scientific data management are faced by both the scientists and the informaticists as described in the previous two sections. Their challenges and needs affect each other. Hence, the proposed system requirements should reflect and address the challenges that both stakeholders face, i.e. from the perspectives of data management and system development. These challenges naturally construct a core wish list which is the focus of this dissertation. This section summarizes this wish list in the form of system requirements. An evaluation plan is then proposed.

2.3.1. Data management requirements

The following lists some of the key system requirements related to data management as would be experienced and tested by the scientist users:

R1. *The system must allow scientific users to manage large and complex datasets for ease of retrieval and organization. Data may be multimedia with metadata. Data may also have complex relationships.*

R2. *The system must support remote data management, allowing multiple users and multiple disciplines to work together.*

R3. *The system must allow scientists to get involved in and contribute to the process of the system design, development and testing process.*

There are many more important characteristics that a LIMS should satisfy. These are well studied in Anderson's JAMIA 2007 paper (Anderson, et al., 2007). However, this thesis's focus is not development of a perfect LIMS. The requirements for scientific users are made simple and sufficient to satisfy only this small key set of requirements.

2.3.2. Development requirements

The following is a list of system requirements for consideration of challenges faced by informaticists:

R4. *The system must keep development time, effort, and cost low.*

R5. *The system should lower the complexity to deal with system evolution.*

Again, this list could be much bigger but these are two key challenges as illustrated by the case study of SIG.

2.3.3. Evaluation plan

The focus of the thesis is on a methodology for developing an advanced LIMS to resolve challenges faced by both the scientists and informaticists. The system will be evaluated against the development requirements above based on the author's critical analysis. Aside from checking things off of the list individually, it is important to see the system working fluidly. This means both informaticists and scientists can work with each other through a life cycle of the application from planning to design, development to deployment, and finally in customization and maintenance.

2.4. Conclusion

Developing an advanced LIMS for scientists to better manage their data is only half of the challenge. The other half is to alleviate the time and effort cost on the part of the informaticists. In considering a solution for LIMS, informaticists have become a necessary stakeholder in addition to scientists. This chapter demonstrates the challenges from Ojemann's and Steven's groups from the University of Washington. These researchers are representative of the targeted audience of this thesis project, which are fast-paced, small to mid-sized university research laboratories with limited IT resources. The challenges call for a new way of developing LIMS to fill in the gaps in which existing solutions do not already fill. This thesis will hereon focus on a frame work for meeting these challenges in LIMS development.

3. EXISTING LIMS SOLUTIONS

A desired LIMS solution would need to meet the needs of both scientific and informatics users. This chapter evaluates existing LIMS solutions based on the system requirements in Section 2.3. The solutions considered range anywhere from off-the-shelf solutions with low technical requirements in Section 3.2 to toolkits that require technical support for customization in Section 3.3. As scientific users' demand for technical power and their desire to have more control over the systems grow, system designs and development naturally shift to a model-driven approach (Section 3.4). For the last two decades, one of the main focuses in LIMS research is increasing efficiency by developing general frameworks and toolkits. This chapter and thesis are focused on the approaches to developing LIMS rather than any specific LIMS requirement.

3.1. Custom Solutions

Each laboratory principal investigator (PI) believes he has a unique information management problem that deserves a custom solution. Custom solutions are most likely to satisfy users, but they are very costly from the perspective of individual labs. From the perspective of a scientific community, they do not encourage potential data sharing.

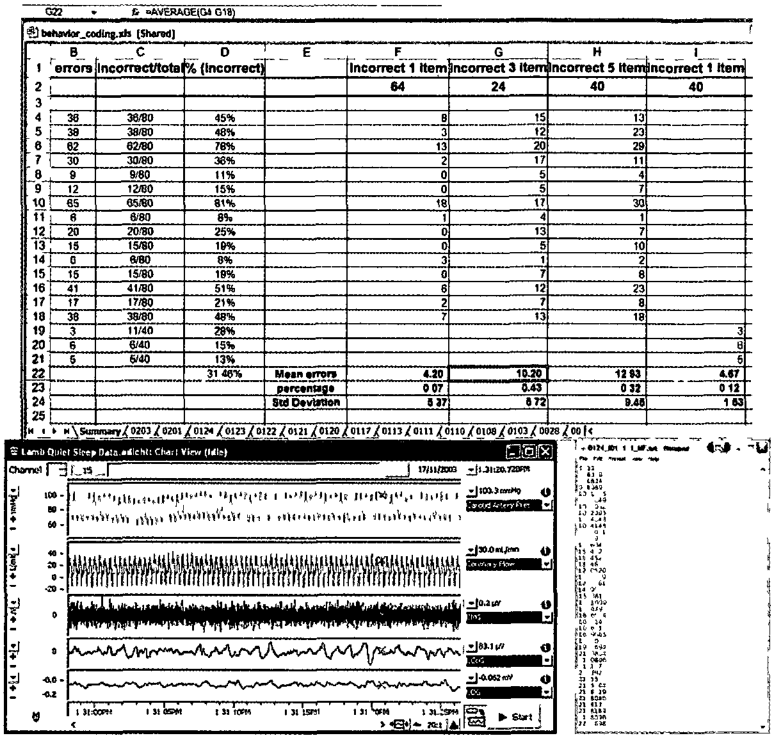
Customized *ad hoc* LIMS are built by software developers with knowledge in database and programming languages, putting together more robust general purpose technology such as web technology and relational databases. While the resulting LIMS meet the requirements of a specific single lab, they cannot be generalized, or adapted for other laboratories, and they cannot evolve quickly. Therefore, they require constant maintenance by a technical expert, which is not commonly available to small research labs. Users have much less control over data, and the maintenance effort is high. The scientists depend on the informaticists for making changes and designs. Highly customized solutions make it difficult to generalize the effort of the informaticists and engineers. The development and maintenance is overly expensive in terms of human expertise and time (Anderson, et al., 2007).

3.2. Off-The-Shelf Solutions

Commercial off-the-shelf solutions (COTS) are the second consideration, because they require the least amount of technical skills on the part of the scientific users. They can be broken into two camps: electronic spreadsheets such as Microsoft Excel and solutions as provided by instrument makers. MS Excel represents general-purpose software that has been repurposed for scientific data management. Instrument makers provide specialized solutions, which cannot be adopted for more general purposes.

3.2.1. Excel spreadsheet

The use of MS Excel spreadsheets has become a state-of-the-art practice in research data management. Excel is highly embraced by the research community, because it is intuitive for users to set up quickly and begin data collection. Spreadsheets are easily



G22 fx =AVERAGE(G4 G18)

behavior_coding.xls [Shared]

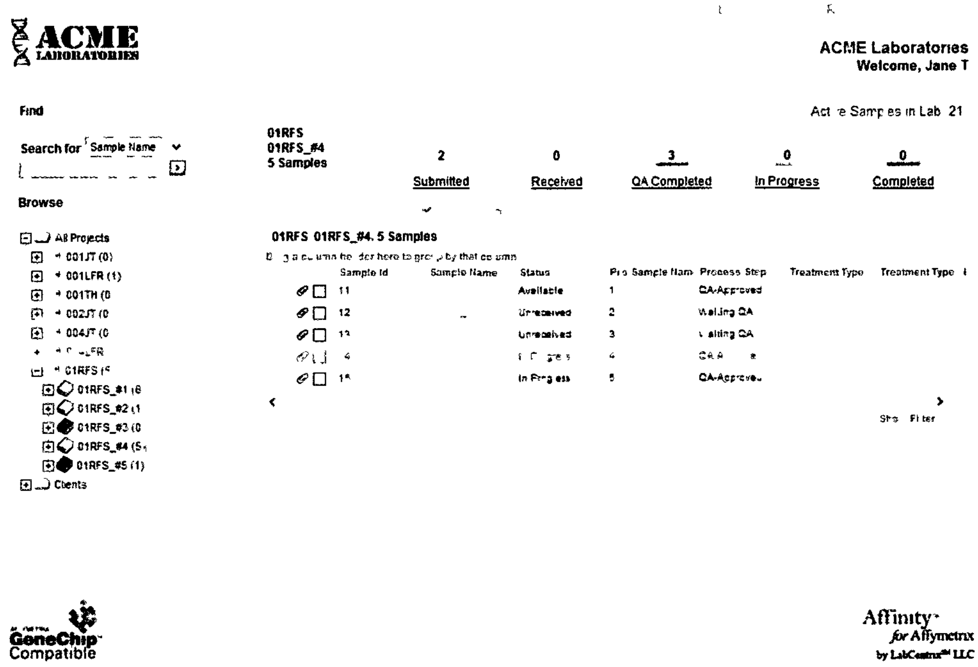
	B	C	D	E	F	G	H	I
1	errors	Incorrect/total	% (Incorrect)		Incorrect 1 item	Incorrect 3 item	Incorrect 5 item	Incorrect 1 item
2					64	24	40	40
3								
4	36	36/80	45%		8	15	13	
5	38	38/80	48%		3	12	23	
6	62	62/80	78%		13	20	29	
7	30	30/80	36%		2	17	11	
8	9	9/80	11%		0	5	4	
9	12	12/80	15%		0	5	7	
10	65	65/80	81%		18	17	30	
11	6	6/80	8%		1	4	1	
12	20	20/80	25%		0	13	7	
13	15	15/80	19%		0	5	10	
14	6	6/80	8%		3	1	2	
15	15	15/80	19%		0	7	8	
16	41	41/80	51%		6	12	23	
17	17	17/80	21%		2	7	8	
18	38	38/80	48%		7	13	18	
19	3	11/40	28%					3
20	6	6/40	15%					6
21	5	5/40	13%					5
22			31 46%	Mean errors	4.20	10.20	12 93	4.67
23				percentage	0 07	0.43	0 32	0 12
24				Std Deviation	5 37	8 72	9.46	1 63
25								

Figure 3.1. A sample screenshot of experiment data captured in an Excel spreadsheet. Each of the worksheets represents data from a subject. Scientists must manually aggregate each worksheet to come up with this summary table. This Excel spreadsheet is inadequate in capturing the neural signal data displayed by Chart on the lower left. The signal data are recorded as a series of timestamps in a text file. The data management complexity is not only time consuming but also error-prone.

adaptable to a domain application and give the scientific users a great sense of control over their data. Excel is easy to learn to use and requires little technical support. However, the complexity of the data, data types, and data volume quickly outgrow what is manageable in Excel, as for example in Figure 3.1, in which time series data from the Ojemann lab are stored in flat files that cannot be easily included in the spreadsheet. The Ojemann lab is an example where concatenating parts of data and ID to form a data file name manually became cryptic and confusing for data management longevity. When interdisciplinary researchers need to work together, they start putting together a cocktail of solutions that do not naturally work together. Excel spreadsheets cannot meet the ubiquitous needs for network accessibility and metadata management (Anderson, et al., 2007).

3.2.2. Instrument maker solutions

The second type of solution is highly customized software provided by instrument makers. Even though these LIMS give users a quick, direct, standardized way of managing data, it is difficult to integrate the LIMS into a real laboratory environment where management of workflow, billing and other data may not be captured. In addition, proprietary data formats limit a lab's access to raw data for developing novel analyses,



ACME LABORATORIES

ACME Laboratories
Welcome, Jane T

Find

Search for Sample Name

Browse

- All Projects
  - 001JT (0)
  - 001LFR (1)
  - 001TH (0
  - 002JT (0
  - 004JT (0
  - LFR
  - 01RFS (5
    - 01RFS_#1 (8
    - 01RFS_#2 (1
    - 01RFS_#3 (0
    - 01RFS_#4 (5
    - 01RFS_#5 (1)
- Clients

Active Samples in Lab 21

01RFS
01RFS_#4
5 Samples

Submitted	Received	QA Completed	In Progress	Completed
2	0	3	0	0

01RFS 01RFS_#4. 5 Samples

Drag a column header here to group by that column

Sample Id	Sample Name	Status	Pro Sample Nam	Process Step	Treatment Type	Treatment Type
11		Available	1	QA-Approved		
12		Unreceived	2	Waiting QA		
13		Unreceived	3	Waiting QA		
14		In Progress	4	QA-Approved		
15		In Progress	5	QA-Approved		

Show Filter

GeneChip Compatible

Affinity for Affymetrix by LabCentrix LLC

Figure 3.2. A sample screenshot of LabCentrix solution for ACME Laboratories. The lab conducts microarray experiments using Affymetrix instruments. LabCentrix LIMS provides a highly complex environment that incorporates data management with lab workflow.

resulting in a fragmented workflow.

To address these fragmentation issues companies, such as LabCentrix (LabCentrix, 2007) or GraphLogic (GraphLogic, 2009), provide LIMS that integrate Affymetrix instrument datasets with other data management needs (Figure 3.2). However, the cost of these solution packages, together with associated consulting services, is beyond what a small academic laboratory can afford. In addition this type of solution is highly complex, and can only satisfy the needs of a narrow niche of labs at the expense of not being general enough to serve widely varied laboratories doing innovative research.

3.3. Customizable toolkit

The previous section demonstrates that off-the-shelf data management tools are limited, expensive, and do not scale well. However, Anderson's study found that while the needs of individual investigators vary across laboratories they also have a great deal of overlap, which could lead to shared LIMS resources and tools. Thus, this section reviews systems that leverage these overlapping needs to create reusable components that can be combined to achieve some amount of customization. These components make up toolkits to be customized by either the scientists themselves or informaticists. Informatics groups such as SIG that provide support to multiple laboratories have long observed design and implementation patterns that could and should be reused (Jakobovits, Rosse, & Brinkley, 2002). Reusing system components leads to lowering the cost of time and resources, and fewer engineers are required to support multiple LIMS. From the perspective of an

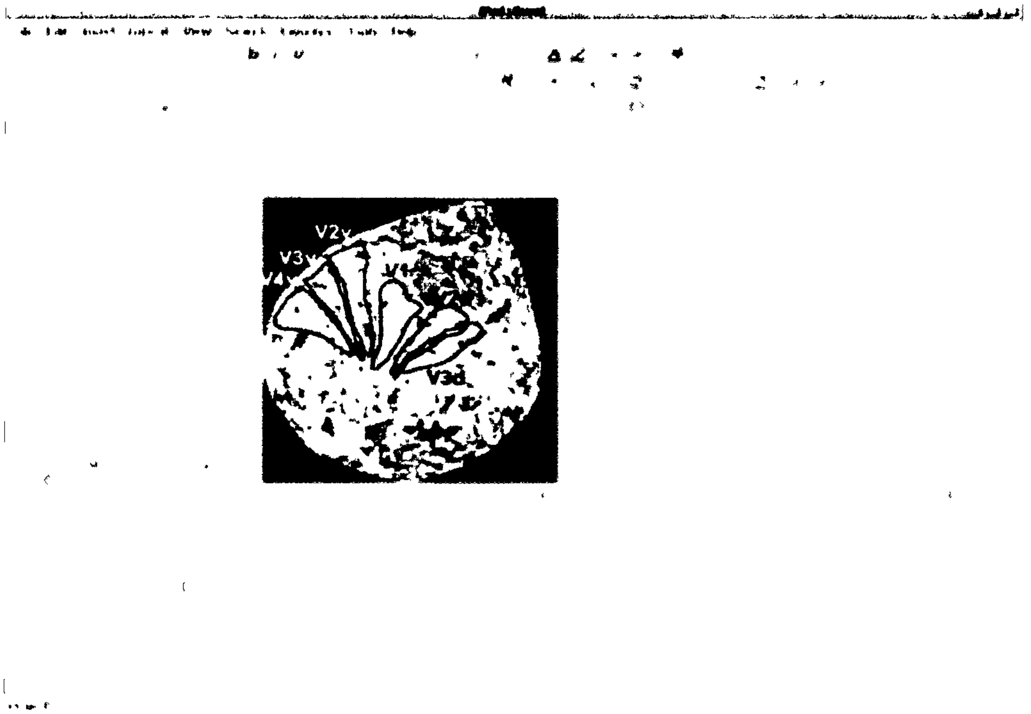


Figure 3.3. A sample screenshop of Ipad. An experiment report shown in the main page is tagged. The tags are organized in a tree structure as shown in the lower left panel.

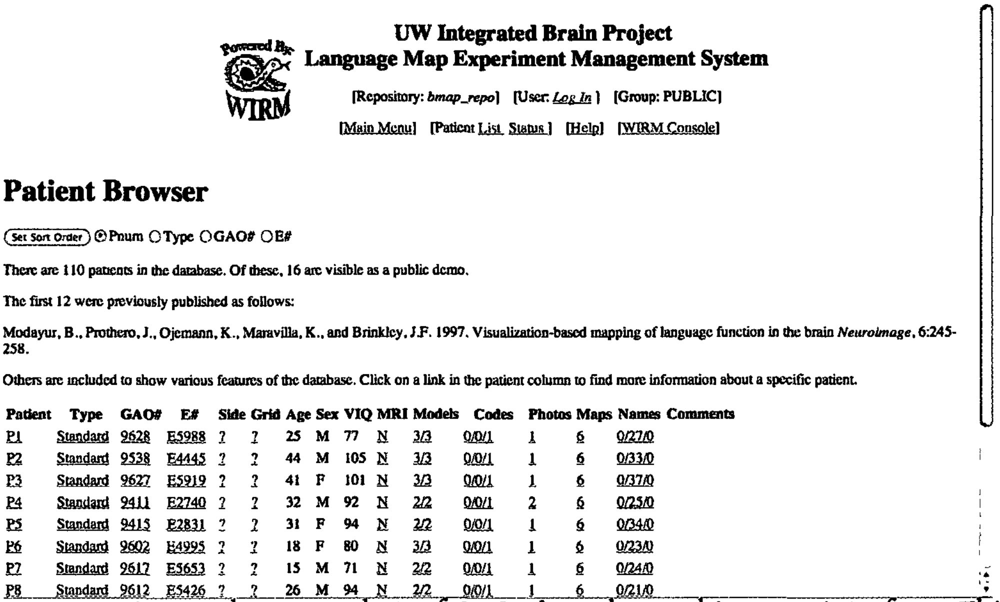
institution, leveraging shared resources is the preferred methodology. Security management of these systems becomes easier as well.

3.3.1. Ipad Electronic laboratory notebook

Ipad Electronic Lab Notebook (Ipad ELN) is unconventional in comparison to most of the laboratory management systems (Ipad, 2010)(Figure 3.3). It allows scientific users to create experiment notebooks as they would in an actual paper notebook. Then it allows the users to tag the different parts of the experiment notes such as hypothesis, result, and task. This tagging feature turns a flat file into a semi-structured file. Users can then exploit the tagged files by performing more effective searches. The obvious benefit of this approach is that the system mimics the scientists' conventional notebook recording with augmented metadata. It can be easily adaptable for scientific users, especially those who are afraid of adopting new technology. Experiment protocols are recorded along with the actual experimental data, making publication and replication of the experiments feasible. The tool allows the users to format and record data in their own way, and by being online, it enables data sharing and collaboration. The major downside to this approach is that it does not provide facilities for large data collection, storage, retrieval and analysis. Without a systematic and machine-readable data structure, this tool cannot support large data manipulation.

3.3.2. WIRM

WIRM (Figure 3.4) was developed by the Structural Informatics Group (SIG) at the University of Washington (Jakobovits, Rosse, & Brinkley, 2002). The framework provides a tool kit that sits in between a custom user interface and advanced open source technology like web servers and relational databases. Specifically, WIRM provides a graphical user interface that allows scientific users to specify their data structures. The middleware automatically generates forms from the data structure information for data entry. Developers create customized code, called wirmlets, which call service APIs such as Web form APIs and database APIs to create custom behavior of the web application. Developers are provided a set of APIs for quickly developing a custom LIMS. This solution fills the gap between COTS and custom solutions. It allows space for developing a highly customized solution while it keeps the cost low by using open-source technology. However, as SIG learned over the years of using WIRM for specific projects such as the Brain Mapper



UW Integrated Brain Project
Language Map Experiment Management System

[Repository: *bmap_repo*] [User: *Log In*] [Group: PUBLIC]

[Main Menu] [Patient List Status] [Help] [WIRM Console]

Patient Browser

(Set Sort Order) ● Pnum ○ Type ○ GAO# ○ E#

There are 110 patients in the database. Of these, 16 are visible as a public demo.

The first 12 were previously published as follows:

Modayur, B., Prothero, J., Ojemann, K., Maravilla, K., and Brinkley, J.F. 1997. Visualization-based mapping of language function in the brain *NeuroImage*, 6:245-258.

Others are included to show various features of the database. Click on a link in the patient column to find more information about a specific patient.

Patient	Type	GAO#	E#	Side	Grid	Age	Sex	VIQ	MRI	Models	Codes	Photos	Maps	Names	Comments
P1	Standard	9628	E5988	?	?	25	M	77	N	3/3	0/0/1	1	6	0/27/0	
P2	Standard	9538	E4445	?	?	44	M	105	N	3/3	0/0/1	1	6	0/33/0	
P3	Standard	9627	E5919	?	?	41	F	101	N	3/3	0/0/1	1	6	0/37/0	
P4	Standard	9411	E2740	?	?	32	M	92	N	2/2	0/0/1	2	6	0/25/0	
P5	Standard	9415	E2831	?	?	31	F	94	N	2/2	0/0/1	1	6	0/34/0	
P6	Standard	9602	E4995	?	?	18	F	80	N	3/3	0/0/1	1	6	0/23/0	
P7	Standard	9617	E5653	?	?	15	M	71	N	2/2	0/0/1	1	6	0/24/0	
P8	Standard	9612	E5426	?	?	26	M	94	N	2/2	0/0/1	1	6	0/21/0	

Figure 3.4. A sample screenshot of WIRM's web graphic user interface. This summary page of experimental subjects is automatically generated by a wirmlet.

Experiment Management System (Brinkley, 2005), increasing requests from the users over the years have evolved the system to becoming highly customizable and difficult to maintain. Evolution became a bottleneck to the system because much of the custom code needs to be evolved in tandem with the data structure changes.

3.3.3. CELO

CELO (Figure 3.5) was also developed at UW SIG as WIRM's successor. It is aimed at quickly creating a database and web application at a low cost. It uses WIRM libraries in addition to its own modules to help users create relational databases through a web front end quickly (Fong & Brinkley, 2006). The database definition can be saved as an XML template file, which can be reused to quickly create new databases by making modifications to the XML file. Different laboratories can share the same database server

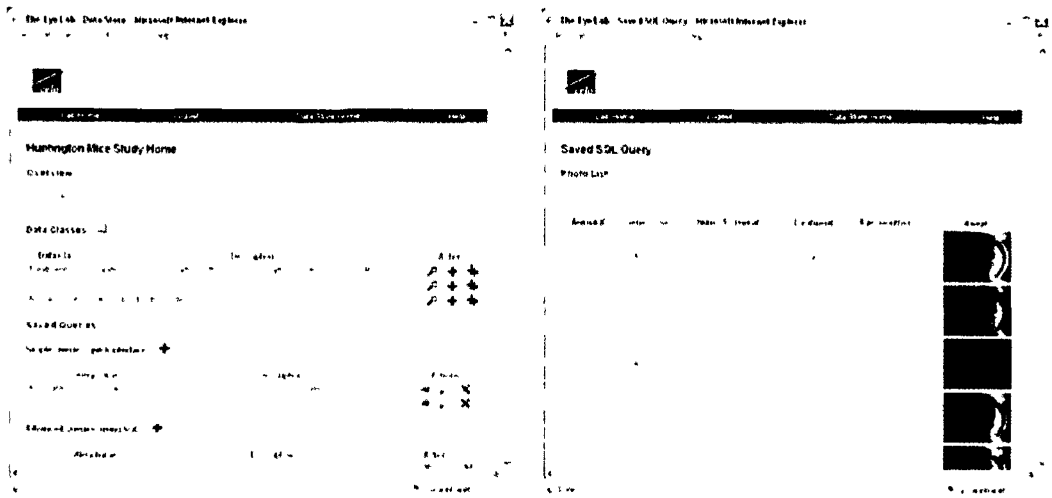


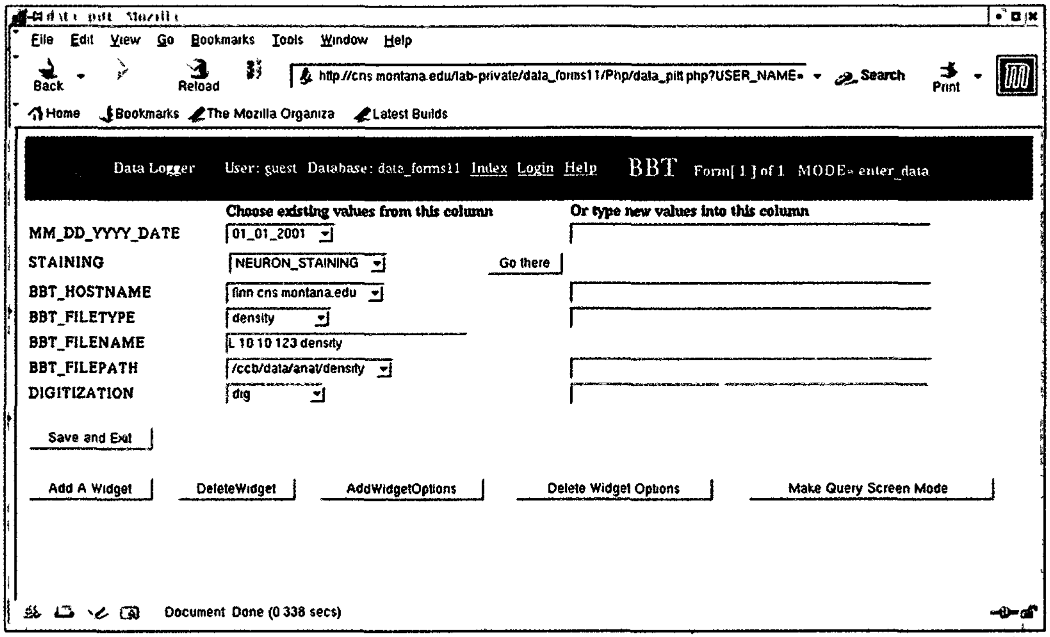
Figure 3.5. Two sample screenshots of CELO's web based user interface. The screenshot on the left shows an administrative page that allows users to manage database objects and saved queries. The screenshot on the right shows an actual data table populated with numeric, textual and graphical data.

but create their own database space. The web application is generic; it can manage data in various databases by inspecting its respective XML database descriptor. However, the database description is basic and limited.

3.3.4. NeuroSys

NeuroSys is another web-based information management system that focuses on solving the data entry problem and reducing database complexity for the users. NeuroSys chooses the semi-structured metadata approach over relational databases, because its developers believe that relational databases are too complex and do not work naturally with auto-generated GUI design (Pittendrigh & Jacobs, 2001).

The users can quickly develop and record data in an *ad hoc* manner through the user interface (Figure 3.6). Behind the scenes, these components are organized in an XML



http//cns montana edu/lab-private/data_forms11/Php/data_pitt php?USER_NAME=

Data Logger User: guest Database: data_forms11 Index Login Help BBT Form[1] of 1 MODE= enter_data

	Choose existing values from this column	Or type new values into this column
MM_DD_YYYY_DATE	01_01_2001	
STAINING	NEURON_STAINING	
BBT_HOSTNAME	finn cns montana edu	
BBT_FILETYPE	density	
BBT_FILENAME	L 10 10 123 density	
BBT_FILEPATH	/ccb/data/anat/density	
DIGITIZATION	dig	

Document Done (0 338 secs)

Figure 3.6. A sample screenshot of NeuroSys. A user can enter data into this data form generated from a pre-existing template. The user can also add or delete widgets from this data entry ad hoc.

structure. The structure does not have to conform to a particular XSD schema. This XML, or parts of the XML that describes data types, can be reused for future data entry. The GUI toolkit is rich, flexible, and expressive. However, what NeuroSys gains in flexibility in metadata would eventually become a performance bottleneck at query time. With lack of key integrity checks as in relational databases, data may tend to be corrupt or incomplete.

3.4. *Model-Driven Approach*

System evolution is inevitable in scientific data management, especially in a small laboratory in which experiment protocols have the shelf life of less than a year. Changes made to data objects and relationships during each evolution can cause a large amount of data engineering and code reengineering. However, this problem is much reduced in solutions with a higher level of metadata abstraction and independence of data model from the program code (Gray, Liu, Nieto-Santisteban, Szalay, DeWitt, & Heber, 2005). A more formal approach to this separation of data model from business logic code is called a model-driven approach (MDA).

A casual definition of a model is adopted here: a limited representation of a system. LIMS models are abstractions of the LIMS system, which encapsulate concepts about the experiments, data management, and laboratory management. Model-driven LIMS allow users to capture their models symbolically or graphically without actual programming. MDA allows software applications to be more flexible and adaptable by capturing what tend to change frequently and in a predicable fashion in the application in a model. The explicit model is interpreted at run-time, and business rules are captured as metadata instead of program code. This allows changes to take place easily in the system. Users can directly change the model without programming (Brown, 2004).

MDA is a powerful concept that was standardized by the Object Management Group(OMG) (MDA, 2010). Model-driven development has a long history in engineering where models are used for simulation, experiment management, and workflow management in a variety of applications (Schmidt, 2006). Lawrence Berkeley Laboratory developed the Object-Protocol Model for developing LIMS for molecular biology applications in 1993 (I-min A. Chen, 1995). At present there are few LIMS that use MDA. This section evaluates two solutions that the author is aware of, Teranode and ManyDesigns Portofino.

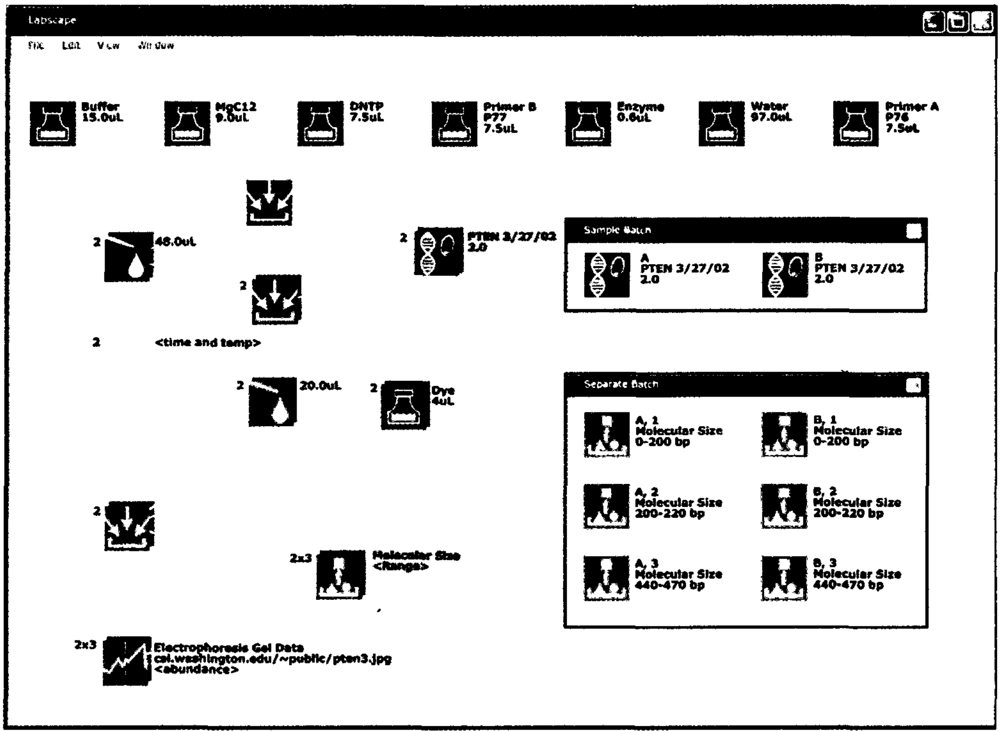
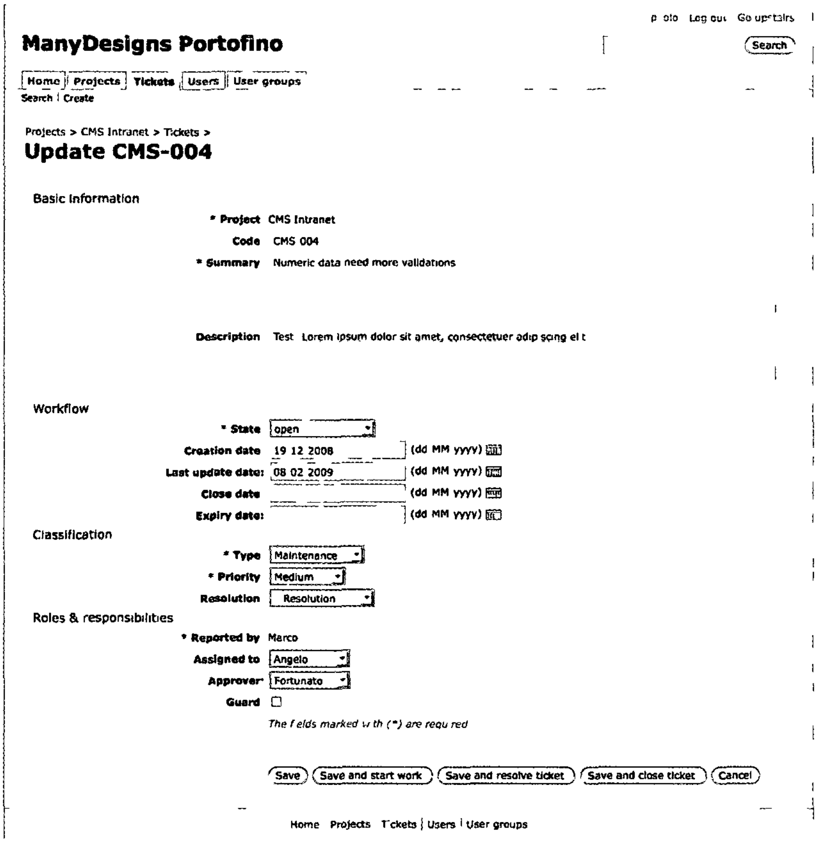

Labscape
File
Edit
View
Window
Buffer 15.0uL
MgC12 9.0uL
DNTP 7.5uL
Primer B P77 7.5uL
Enzyme 0.6uL
Water 97.0uL
Primer A P76 7.5uL
2 48.0uL
2
2 <time and temp>
2 PTEN 3/27/02 2.0
Sample Batch
A PTEN 3/27/02 2.0
B PTEN 3/27/02 2.0
2 20.0uL
2 Dye 4uL
Separate Batch
A, 1 Molecular Size 0-200 bp
B, 1 Molecular Size 0-200 bp
A, 2 Molecular Size 200-220 bp
B, 2 Molecular Size 200-220 bp
A, 3 Molecular Size 440-470 bp
B, 3 Molecular Size 440-470 bp
2
2x3 Molecular Size <Range>
2x3 Electrophoresis Gel Data cel.washington.edu/~public/pten3.jpg <abundance>

Figure 3.7. A sample screenshot of Teronode's visual experimental protocol design environment. Each node in the graph denotes a data entry step or experimental step. Paths between nodes denote workflow sequence. They may contain data transformation and calculations.

3.4.1. Teranode

Teranode is a Seattle-based startup (Teranode, 2010). The LIMS is built on top of previous research in LabScape spearheaded by one of its co-founders Larry Arnestein (Arnstein, Hung, Franza, & Zhou, 2002; Arnstein, et al., 2002). The system offers tools for experiment data acquisition and automation. It also provides a model design environment that allows informaticists or scientists to design experiment protocols. The system is open and dynamic, and can be quickly integrated to work with different instrument platforms for automatic high throughput data acquisition.



Projects > CMS Intranet > Tickets >

Update CMS-004

Basic Information

* Project CMS Intranet
Code CMS 004
* Summary Numeric data need more validations

Description Test Lorem ipsum dolor sit amet, consectetuer adip scing el t

Workflow

* State open
Creation date 19 12 2008 (dd MM yyyy)
Last update date: 08 02 2009 (dd MM yyyy)
Close date (dd MM yyyy)
Expiry date: (dd MM yyyy)

Classification

* Type Maintenance
* Priority Medium
Resolution Resolution

Roles & responsibilities

* Reported by Marco
Assigned to Angelo
Approver Fortunato
Guard □

The f elds marked w th (*) are requ red

Figure 3.8. A sample screen of ManyDesign's data update form. The form is automatically generated based on the CMS model definition.

Teranode developed its own XML-based modeling language called Visual Language of Experimentation, or VLX. VLX allows users to represent, annotate, and share information about complex experiment data objects, relationships and workflow. The icon-based modeling environment (Figure 3.7) allows testing and debugging the model with ease. Ultimately, the VLX model is automated and executed in real time in an experiment coordinating data entry, lab workflow, and report generation. Recorded data is stored in a XML database. The system suffers from query and retrieval efficiency when the dataset becomes large. Like NeuroSys, Teranode gains flexibility and expressivity by using XML.

However, Teranode locks down the protocol from changes before scientists start using it for data entry. Teranode is a generalized solution that has the promise to lower cost and time of development, while providing ease for data management in a complex laboratory setting. It aims to serve large production pharmaceutical laboratories and it is not open source. As of 2010, the company has shifted focus away from their LIMS development module and the fate of the company is unclear.

3.4.2. ManyDesigns Portofino

Portofino is an open source solution developed by another privately owned company in Italy called ManyDesigns (ManyDesigns, 2010). Unlike Teranode, its solution is designed for more general purpose use. Little is known in publication about the product but from what can be implied from their website, Portofino works on top of a model that defines a web application. It is developed for much more general purpose applications than just LIMS. This model contains classes, attributes, relationships, user permission and workflow. The model is transformed to create a relational database. The web-based GUI is auto-generated for data entry, browsing and reporting (Figure 3.8). To use Portofino, users download Portofino and install on their own web server. The server application is configured to work with various database systems such as Oracle, Microsoft SQL Server, PostgreSQL and MySQL. Portofino's greatest advantage is its ability to change its Data Definition Language (DDL) in real time. Unfortunately, information on the kind of changes and how it treats existing data through this schema evolution is not clear from the lack of English publications and their website.

3.5. Conclusions

This section provides a summary comparison between all of the four categories of LIMS described in this chapter and weighs them against the requirements listed in 2.3, concluding that MDA is superior to others.

3.5.1. Summary of existing solution and approaches

Four categories of LIMS solutions are reviewed in this chapter with a focus on the five requirements listed in Section 2.3. The five requirements are captured into the column headings in Figure 3.9. *R1* (data management features), refers to the system's ability to manage large and complex data types and relationships. *R2* (multi-user remote access) is the system's ability to allow multiple users from a research team to access and manage data simultaneously without version control or synchronization issues. *R3* (scientist user involvement) refers to the level of user involvement in the design and development of their LIMS. *R4* requires lowering the development time and technical cost. *R5* refers to the ease of system evolution as a result of data model changes. The rows in Figure 3.9 are the four categories of existing solutions reviewed in this chapter. Each system is given a score for each of the requirements. The available scores 1, 2, and 3 correspond to low, medium and high respectively. Finally, the scores are summed for each solution group for comparison. This section summarizes how the descriptions in the prior sections contribute to the scores.

	R1 Data management features	R2 Multi-user remote access	R3 Scientist user involvement	R4 Low development time and technical cost	R5 Ease of system evolution	Total
S1: Custom solutions	3	3	1	1	1	9
S2: COTS (Excel/Affymetrix)	2 (1/3)	2 (1/3)	2 (3/1)	1 (1/1)	2 (3/1)	9
S3: Tool kits	3	3	1	2	1	10
S4: Model-driven	2	3	2	3	3	13

Figure 3.9. Four solution categories, custom solutions, COTS, tool kits, and model-driven systems, are scored 1 (low), 2 (medium) or 3 (high) for each of the requirements listed in Chapter 2. Their totals are compared. Model-driven solutions are the leader.

S1: Custom solutions can provide high user satisfaction in terms of data management and many of them already adopt a network enabled system. Users contribute little to the development of the system as it requires high level of engineering expertise. The cost is high (hence score 1 for low-cost) and this highly customized solution lacks generality for ease of change. One may observe that tool feature satisfaction is achieved at the compromise of reusability and cost.

S2: COTS solutions described in this chapter range widely from general-purposed and low cost Excel to expensive specialized instrument maker solutions. Both of these are considered to come readily usable by the scientific users. The two solutions are at polarity with each other for four out of five requirements. They are both costly with the difference being that Excel is expensive in terms of manual user labor to set it up and maintain it in the long run as opposed to the actual cost of purchasing a specialized system. A correlation exists between the ease of change and generalizability, which are both inversely related to functionalities of a system. In either case, the users need to spend a lot more energy to adapt the system either technically or culturally into an existing work environment.

S3: Toolkit solutions are akin to custom solutions with an aim to lower development and maintenance cost through re-usable components. Fewer engineers are required to support multiple LIMS. From the perspective of an institution, it is much more preferred to leverage shared resources between its laboratories. Security management of these systems becomes easier as well. However, tool kit solutions depend highly on skilled engineers. Evolution of a system due to data model changes is labor intensive and complex. Additionally, scientists are further away from the tools they are familiar with. The process of developing the system can be unsupported and frustrating.

S4: Model-driven solutions combine the advantages of the toolkit solutions with added focus on reusability, change management, and flexibility. In comparison to S3, they further decrease development cost and increase scientist users' engagement in the design and development process of a LIMS. The model integrates the reusable components of tool kits solutions (S3) into a declarative abstraction of a LIMS system, making it easy for fast prototyping, especially for non-technical users. Several research studies have shown that making changes to an information system is easier by using the MDA approach. (Hick & Hainaut, 2003; Dominguez, Lloret, & Rubio, 2002; Estrella, Kovacs, Goff, McClatchey, & Toth, 2001).

3.5.2. The Seedpod Model Driven Approach

The general trend in LIMS development and research is focused on lowering development cost and time by generalizing some aspects of the system. The more pressing question at hand is how to lower the cost and time of making changes to an existing system. As shown in Figure 3.9, model-driven LIMS solutions seem to have the answer to that question. However, there are only two examples of MDA-based solutions to LIMS development that the author is aware of. These solutions are not readily accessible, either because they are no longer supported or because they are not well documented. In addition neither is based on a rich knowledge model as represented in ontology. Thus, the remaining chapters describe and evaluate my own MDA solution to LIMS development, which is implemented in the knowledge model based Seedpod system.

4. SEEDPOD (A CASE STUDY)

This chapter describes Seedpod, an implemented scientific data management system which demonstrates the model driven approach (MDA) described in the previous chapter. Seedpod attempts to abstract the complexity of a data management system from the perspectives of two primary user groups: scientific researchers and informaticists. The chapter unpacks the architectural design and technical implementation details of Seedpod. The primary focus is to show how this MDA approach to implementing LIMS helps to a) separate design from implementation technology, b) hide technical complexity to keep the focus on domain problems, and c) maintain a certain level of scalability.

Section 4.1 describes the overall architectural design of the system. Section 4.2 to 4.5 describes the three main components in detail: model, transform, and application engine. Finally, Section 4.7 describes how scientific researchers and informaticists are intended to interact with Seedpod.

4.1. Model-Driven Architecture

Seedpod implements a model-driven architecture. There are three major components: 1) model, 2) transformation, and 3) LIMS web application (webapp) (Figure 4.1). The platform-independent model (PIM) represented in Protégé serves as an abstraction to the LIMS. The transformation component translates the PIM to platform-specific models (PSM), such as SQL in Seedpod, which can be executed directly. Unlike other MDA systems, Seedpod does not generate platform-specific code. The three

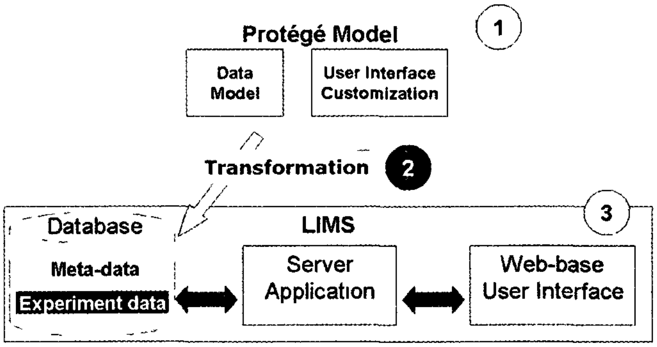


Figure 4.1. Seedpod architecture with three components: 1) Protégé model, 2) transformation engine, and 3) web-based LIMS application.

components are not tightly coupled, i.e. they can be developed and evolved asynchronously.

The PIM (1 in Figure 4.1) is an integrated representation of a LIMS declaratively represented using Protégé. It includes a domain-specific data model describing the entities and relationships that the scientific users wish to manage. It also includes an application model describing properties for customizing the look and feel of the LIMS web-based user interface.

The second component (2 in Figure 4.1) is a transformation program that automatically translates the Protégé model into a relational model for the backend relational database. The database stores scientific data and meta-data on the mapping of concepts between the two models. This meta-data describes a subset of the original Protégé model used by the LIMS application. The transformation engine is non-domain specific, while both the Protégé model and the relational model are domain-specific.

The third component is the LIMS application engine (3 in Figure 4.1). It includes the server application, relational database, and a web-based graphical user interface (GUI).

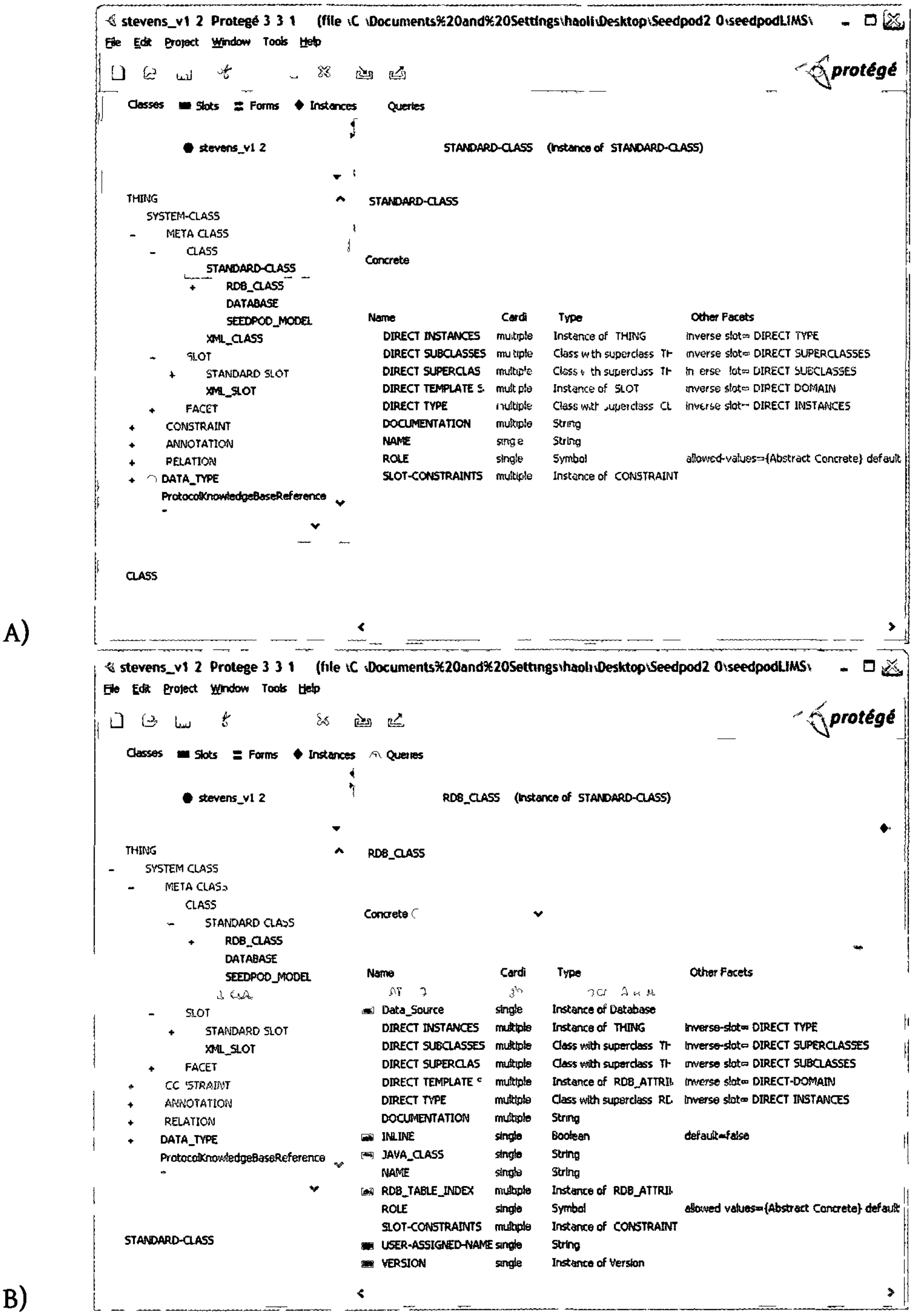
	Platform	Domain
Protégé Model	Platform-independent	Domain-specific
Transformation Engine	Platform-dependent	Non-domain-specific
Relational Database	Platform-dependent	Domain-specific
Server Application Engine	Platform-dependent	Non-domain-specific

Figure 4.2. Seedpod components' dependency on their implementation platform and domain.

Similar components would be found in a conventional web-based application with a database backend. The database stores the experiment data and meta-data (or LIMS model). The server application queries the database regarding the model, retrieves and stores the experiment data, and finally, generates dynamic web pages for users. As mentioned previously, the web server application code is not auto-generated from the model through a transformation process. The application is non-domain dependent. Figure 4.2 summarizes the components and whether they contain domain specific information.

4.2. Modeling Using Protégé

Seedpod uses Protégé for modeling. Protégé provides a graphical user interface that allows users to model a domain with a set of representation constructs such as classes, slots and facets. Behind the scene, the models can be saved in various formats such as Protégé projects (*.pprj),* XML, relational databases, or RDF. The models can also be programmatically accessed through a JAVA API.


A)
stevens_v1 2 Protegé 3 3 1 (file \C \Documents%20and%20Settings\haoli\Desktop\Seedpod2 0\seedpodLIMS\
File Edit Project Window Tools Help
protégé
Classes Slots Forms Instances Queries
stevens_v1 2
THING
SYSTEM-CLASS
META CLASS
CLASS
STANDARD-CLASS
RDB_CLASS
DATABASE
SEEDPOD_MODEL
XML_CLASS
SLOT
STANDARD SLOT
XML_SLOT
FACET
CONSTRAINT
ANNOTATION
RELATION
DATA_TYPE
ProtocolKnowledgeBaseReference
CLASS
B)
stevens_v1 2 Protege 3 3 1 (file \C \Documents%20and%20Settings\haoli\Desktop\Seedpod2 0\seedpodLIMS\
File Edit Project Window Tools Help
protégé
Classes Slots Forms Instances Queries
stevens_v1 2
THING
SYSTEM CLASS
META CLASS
CLASS
STANDARD CLASS
RDB_CLASS
DATABASE
SEEDPOD_MODEL
SLOT
STANDARD SLOT
XML_SLOT
FACET
CONSTRAINT
ANNOTATION
RELATION
DATA_TYPE
ProtocolKnowledgeBaseReference
STANDARD-CLASS

A)

STANDARD-CLASS (instance of STANDARD-CLASS)

STANDARD-CLASS

Concrete

Name	Cardi	Type	Other Facets
DIRECT INSTANCES	multiple	Instance of THING	inverse slot= DIRECT TYPE
DIRECT SUBCLASSES	mu tiple	Class w th superclass TH	inverse slot= DIRECT SUPERCLASSES
DIRECT SUPERCLAS	multiple	Class w th superclass TH	in erse lot= DIRECT SUBCLASSES
DIRECT TEMPLATE S	mult ple	Instance of SLOT	inverse slot= DIRECT DOMAIN
DIRECT TYPE	multiple	Class with superclass CL	inverse slot= DIRECT INSTANCES
DOCUMENTATION	multiple	String	
NAME	sing e	String	
ROLE	single	Symbol	allowed-values={Abstract Concrete} default
SLOT-CONSTRAINTS	multiple	Instance of CONSTRAINT	

B)

RDB_CLASS (instance of STANDARD-CLASS)

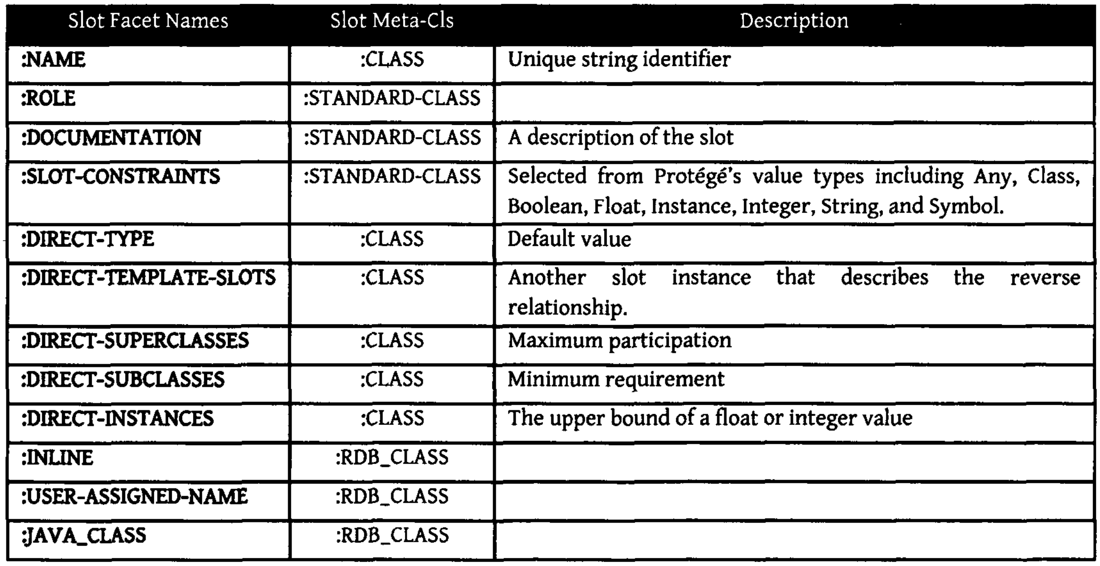
RDB_CLASS

Concrete

Name	Cardi	Type	Other Facets
Data_Source	single	Instance of Database	
DIRECT INSTANCES	multiple	Instance of THING	inverse-slot= DIRECT TYPE
DIRECT SUBCLASSES	multiple	Class with superclass TH	inverse-slot= DIRECT SUPERCLASSES
DIRECT SUPERCLAS	multiple	Class with superclass TH	inverse slot= DIRECT SUBCLASSES
DIRECT TEMPLATE S	multiple	Instance of RDB_ATTRIB	inverse slot= DIRECT-DOMAIN
DIRECT TYPE	multiple	Class with superclass RD	inverse slot= DIRECT INSTANCES
DOCUMENTATION	multiple	String	
INLINE	single	Boolean	default=false
JAVA_CLASS	single	String	
NAME	single	String	
RDB_TABLE_INDEX	multiple	Instance of RDB_ATTRIB	
ROLE	single	Symbol	allowed values={Abstract Concrete} default
SLOT-CONSTRAINTS	multiple	Instance of CONSTRAINT	
USER-ASSIGNED-NAME	single	String	
VERSION	single	Instance of Version	

Figure 4.3. Screenshots from Protégé showing the differences between the Protégé provided basic meta-class *:STANDARD-CLS* (A) and the Seedpod meta-class *:RDB_CLASS* (B). Extensions such as this allow customized domain specific modeling to take place easily.

One of the advantages of using Protégé is that its meta-model can be extended for modeling richer domain specific knowledge (Noy, Sintek, Decker, Crubezy, Fergerson, & Musen, 2001; Gitzel & Korthaus, 2004). Seedpod expands upon the standard Protégé meta-model by including *:RDB_CLS* and *:RDB_SLOT.* These new meta-classes inherit from the standard system classes *:STANDARD-CLS* and *:STANDARD-SLOT* respectively. The custom meta-classes are used exclusively as the default meta-classes in Seedpod. They allow users to say more about a particular class (Figure 4.3). Figure 4.4 and Figure 4.5 show listings of all the facets of classes *:RDB_CLASS* and *:RDB_SLOT,* respectively. Some of the facets are inherited from *:STANDARD-CLASS* and :*STANDARD-SLOT* while many are custom added for Seedpod (they have *:RDB_CLASS* and *:RDB_ATTRIBUTE* as their meta-class type in the respective figures).

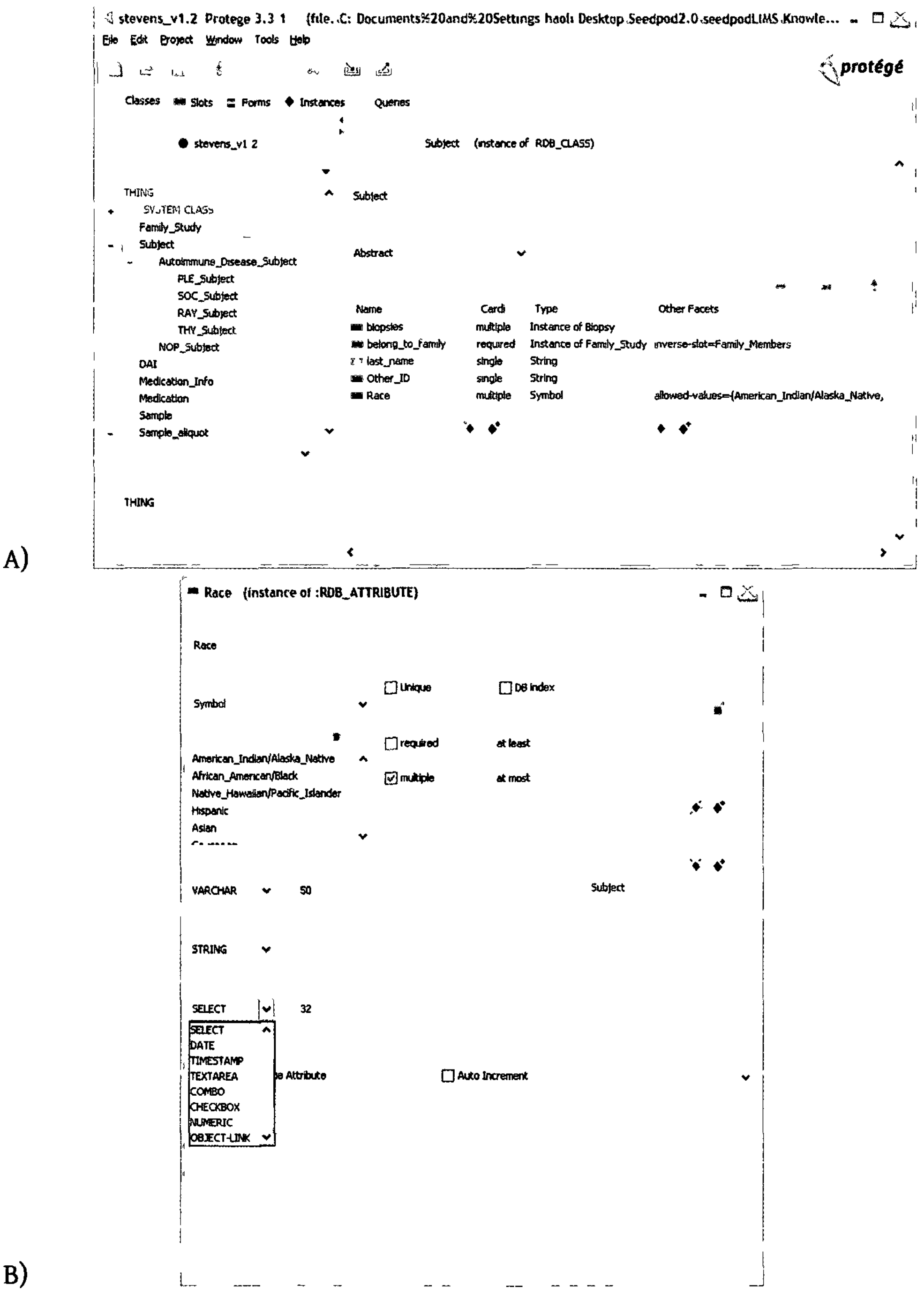


Slot Facet Names	Slot Meta-Cls	Description
:NAME	:CLASS	Unique string identifier
:ROLE	:STANDARD-CLASS	
:DOCUMENTATION	:STANDARD-CLASS	A description of the slot
:SLOT-CONSTRAINTS	:STANDARD-CLASS	Selected from Protégé's value types including Any, Class, Boolean, Float, Instance, Integer, String, and Symbol.
:DIRECT-TYPE	:CLASS	Default value
:DIRECT-TEMPLATE-SLOTS	:CLASS	Another slot instance that describes the reverse relationship.
:DIRECT-SUPERCLASSES	:CLASS	Maximum participation
:DIRECT-SUBCLASSES	:CLASS	Minimum requirement
:DIRECT-INSTANCES	:CLASS	The upper bound of a float or integer value
:INLINE	:RDB_CLASS	
:USER-ASSIGNED-NAME	:RDB_CLASS	
:JAVA_CLASS	:RDB_CLASS	

Figure 4.4. Listing of facets that describe customized Seedpod meta-slot class :*RDB_CLASS.*

Slot Facet Names	Domain Meta-Cls	Description
:NAME	:SLOT	Unique string identifier
:DIRECT-DOMAIN	:SLOT	
:DOCUMENTATION	:STANDARD-SLOT	A description of the slot
:SLOT-VALUE-TYPE	:SLOT	Selected from Protégé's value types including Any, Class, Boolean, Float, Instance, Integer, String, and Symbol.
:SLOT-DEFAULTS	:STANDARD-SLOT	Default value
:SLOT-INVERSE	:STANDARD-SLOT	Another slot instance that describes the reverse relationship.
:SLOT-MAXIMUM-CARDINALITY	:STANDARD-SLOT	Maximum participation
:SLOT-MINIMUM-CARDINALITY	:STANDARD-SLOT	Minimum requirement
:SLOT-NUMERIC-MAXIMUM	:STANDARD-SLOT	The upper bound of a float or integer value
:SLOT-NUMERIC-MINIMUM	:STANDARD-SLOT	The lower bound of a float or integer value
:USER-ASSIGNED-NAME	:RDB_ATTRIBUTE	A better display name for GUI
:DATABASE-INDEX	:RDB_ATTRIBUTE	A flag for whether the slot should be indexed in the database
:DATABASE-TYPE	:RDB_ATTRIBUTE	Value type for storage in a relational database. Options include Integer, Varchar, Boolean, Character, Numeric, Text, Date, Time, Timestamp
:DATABASE-TYPE-PARAMETER	:RDB_ATTRIBUTE	Parameter to database type. For example, length of varchar.
:INLINE_ATTRIBUTE	:RDB_ATTRIBUTE	A flag which sets an instance type slot to be in-lined. For example, slot instance of class Date has three in-lined attributes: year, month, and day.
:PERMISSION	:RDB_ATTRIBUTE	Field level permission setting. (Not implemented)
:UNIQUE	:RDB_ATTRIBUTE	A flag for whether a value can only exists once in the database.
:UNIT	:RDB_ATTRIBUTE	Name of measurement. For example, meters, inches.
:VALUE-EXPRESSION	:RDB_ATTRIBUTE	Formula or logic for calculating the value of this slot. *(Not implemented)*
:VIEW-SEQUENCE	:RDB_ATTRIBUTE	Sequence number for the display of this slot in the web-based GUI.
:FORM-WIDGET	:RDB_ATTRIBUTE	HTML widget for data input. The allowed values depends on implemented widget plug-ins in the web application.
:FORM-WIDGET-PARAMETER	:RDB_ATTRIBUTE	A naïve way for inputting parameters to the form widget.
:VIEW-WIDGET	:RDB_ATTRIBUTE	HTML widget for data display
:VIEW-WIDGET-PARAMETER	:RDB_ATTRIBUTE	A naïve way for inputting parameters to the widget.
:DIRECT-TYPE	:SLOT	Slot meta-class. (Ignored. Seedpod only uses slots that are instances of :RDB_SLOT or children of :RDB_SLOT)
:ASSOCIATED-FACET	:STANDARD-SLOT	(Ignored for Seedpod)
:DIRECT-SUBSLOTS	:STANDARD-SLOT	(Ignored for Seedpod)
:DIRECT-SUPERSLOTS	:STANDARD-SLOT	(Ignored for Seedpod)
:SLOT-CONSTRAINTS	:STANDARD-SLOT	(Ignored for Seedpod)
:SLOT-VALUES	:STANDARD-SLOT	(Ignored for Seedpod)

Figure 4.5. Listing of facets that describe customized Seedpod meta-slot class :*RDB_SLOT*. The customized facets are added to give more information about a slot.



A)

stevens_v1.2 Protege 3.3 1 (file..C: Documents%20and%20Settings haoli Desktop Seedpod2.0 seedpodLIMS Knowle...

File Edit Project Window Tools Help

Classes Slots Forms Instances Queries

● stevens_v1 2

Subject (instance of RDB_CLASS)

THING
- SYSTEM CLASS
- Family_Study
- Subject
  - Autoimmune_Disease_Subject
    - PLE_Subject
    - SOC_Subject
    - RAY_Subject
    - THY_Subject
  - NOP_Subject
- DAI
- Medication_Info
- Medication
- Sample
- Sample_aliquot

THING

Subject

Abstract

Name	Cardi	Type	Other Facets
biopsies	multiple	Instance of Biopsy	
belong_to_family	required	Instance of Family_Study	inverse-slot=Family_Members
last_name	single	String	
Other_ID	single	String	
Race	multiple	Symbol	allowed-values={American_Indian/Alaska_Native,

Race (instance of :RDB_ATTRIBUTE)

Race

Symbol

□ Unique □ DB Index

American_Indian/Alaska_Native
African_American/Black
Native_Hawaiian/Pacific_Islander
Hispanic
Asian

□ required at least

☑ multiple at most

VARCHAR 50

Subject

STRING

SELECT 32

SELECT
DATE
TIMESTAMP
TEXTAREA
COMBO
CHECKBOX
NUMERIC
OBJECT-LINK

...e Attribute □ Auto Increment

B)

Figure 4.6. A) An example of the Protégé modeling environment. This also shows an example of class inheritance modeled in Protégé. Parent abstract class *Subject* is specialized to several concrete classes (e.g. *PLE_Subject*, *SOC_Subject*), each with distinct slots. **B)** A screen shot of a slot modeling form. Note that the slot meta-facets that were custom added such as Form widget, DB type, are available to the modeler.

Several knowledge-based approaches to database design exist (Noah & Lloyd-Williams, 1995). An integrated model of data objects and the LIMS application is necessary (Goodman, Rozen, Stein, & Smith, 1998; I-min A. Chen, 1995). The majority of the facets describe data elements. For example, *:DATABASE-TYPE* (Figure 4.5) allows the modeler to specify whether a slot should be implemented as "*DATETIME*" or "*VARCHAR*" or "*TEXT*" in the relational database. In this case, this newly added facet clarifies an example of model impedance between Protégé and RDB.

Additional facets are created to describe the look-and-feel in the LIMS application. For example, "*:FORM-WIDGET*" allows the user to specify the plug-in widgets that are available in the web application. An example of how some of the slot facets are applied can be seen in definition of slot *race* in Figure 4.6.B.

Protégé is used in software applications for domain ontology management (Musen, 1998). Separating the domain knowledge eases application maintenance. Users can create domain-specific model classes by creating instances of these meta-classes such as classes *Subject* and *Medication*. Each class is further described by a set of slots, or attributes. For example, the *Subject* class is described by slots such as *last_name*, *race*, *ID*, etc. Modeling classes and slots are demonstrated with examples in Figure 4.6. The idea is for scientific researchers, who design experiments and have domain knowledge of the data model, to describe the data objects inside of Protégé using its frame-based modeling environment, which is similar but richer than the more familiar object-oriented (OO) modeling environment (Noy & McGuinness, 2001).

The OO-like approach to modeling relationships may be more intuitive than normalized relational modeling for naïve modelers such as scientists. In the Protégé environment, relationships between classes are represented by slots of instance types. A relationship is directional with a from-class and a to-class. The from-class contains an instance type slot that is of type to-class. The relationship is named by the slot. The cardinality of the relationship is also defined by the slot. For example, class *Subject* is related to class *Family* through slot *belong_to_family*. Note that an inverse relationship, from *Family* to *Subject* is also defined by slot *family_members* (Figure 4.6.A). The significance of the inverse relationship representation is discussed in depth in Chapter 5.

Inheritance relationships can be modeled in Protégé. For example, one can create an abstract class called *Subject*. An abstract class differs from a concrete class in that it cannot have actual instance data. By using inheritance, the user can create more specialized *Subject* classes, such as *NOP_Subject* (control subject), with some slots inherited from *Subject* and customized slots that distinguish it from other types of Subjects such as *PLE_Subject, SOC_Subject* in Figure 4.6.A.

4.3. Model Transformation

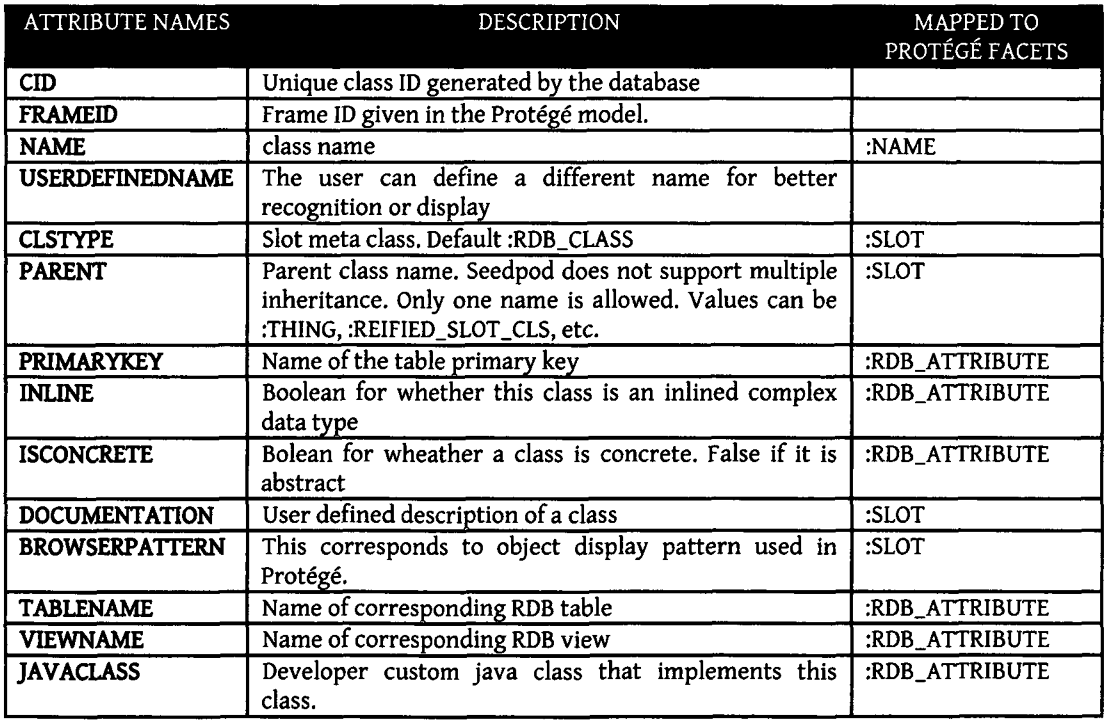
Protégé stores data in an entity-attribute-value triple fashion, which is inefficient for large data set retrieval assuming the data are not highly sparse (Entity-attribute-value model, 2010; Nadkarni, Marenco, Chen, Skoufos, Shepherd, & Miller, 1999). An object-relational style database is used to store data in Seedpod for efficient data access and storage. Thus, it is necessary to transform the Protégé model into a relational model. An automatic transformation is developed. Such an approach has been shown beneficial in gene sequence data (Rubin, Shafa, Oliver, Hewett, & Altman, 2002). Chapter 5 describes in detail the theory, implementation and the outcome for this automated method. The transformation program is what allows Seedpod to leverage both Protégé's design GUI environment and a robust relational database that may have been prohibitive for naïve users. Running the transformation returns consistent predictable results. The resulting database definition is in a text file which can be examined before it is used to create a database. The resulting SQL conforms to the SQL-99 standard, which means it should be executable in any relational database management system that implements the standard. The content of the transformation database definition consists of a database schema for storing scientific data and meta-data tables populated with mappings between the Protégé schema elements and RDB schema elements. This mapping meta-data becomes the brain for the server-based application.

4.4. Relational Database

The transformation step results in a database definition written in SQL which can be used directly to create a database. Each seedpod database instance has two components: data and meta-data. The database schema is described in detail as a result of the transformation method in Chapter 5. This section summarizes the characteristics of the database. Seedpod uses a PostgreSQL database to store its data.

4.4.1. Data tables and views

Each Seedpod data model is different depending on the specific domain application (model-specific). The database tables are mostly normalized with the exception of tables storing inherited objects (see horizontal vs. vertical fragmentation discussion in Section 5.3.2). The schema is optimized for insertion, editing, and retrieval of objects defined by Protégé classes. Furthermore, views are pre-constructed for ease of querying one object at a time without users having to deal with queries with joins. They also enable users to query objects in an inheritance tree by the parent type. This approach is similar to an Object-Relational database (Liu, Orlowska, & Li, 1997). For example, given an inheritance tree of wine varietals, querying for instances of *wine* can return instances of *pinot*, *merlot*, etc.



ATTRIBUTE NAMES	DESCRIPTION	MAPPED TO PROTÉGÉ FACETS
CID	Unique class ID generated by the database	
FRAMEID	Frame ID given in the Protégé model.	
NAME	class name	:NAME
USERDEFINEDNAME	The user can define a different name for better recognition or display	
CLSTYPE	Slot meta class. Default :RDB_CLASS	:SLOT
PARENT	Parent class name. Seedpod does not support multiple inheritance. Only one name is allowed. Values can be :THING, :REIFIED_SLOT_CLS, etc.	:SLOT
PRIMARYKEY	Name of the table primary key	:RDB_ATTRIBUTE
INLINE	Boolean for whether this class is an inlined complex data type	:RDB_ATTRIBUTE
ISCONCRETE	Bolean for wheather a class is concrete. False if it is abstract	:RDB_ATTRIBUTE
DOCUMENTATION	User defined description of a class	:SLOT
BROWSERPATTERN	This corresponds to object display pattern used in Protégé.	:SLOT
TABLENAME	Name of corresponding RDB table	:RDB_ATTRIBUTE
VIEWNAME	Name of corresponding RDB view	:RDB_ATTRIBUTE
JAVACLASS	Developer custom java class that implements this class.	:RDB_ATTRIBUTE

Figure 4.7. Listing of attributes in meta-data table *:RDB_CLASS.*

Attribute Name	Mapped to Protégé Slot Facet (See Figure 4.5 Column 1)	Comments
aid		Unique attribute ID generated by the database
frameID		Frame ID given in the Protégé model.
domainCls	:DIRECT-TYPE	Containing class of the slot
name	:NAME	Name of the slot
userDefinedName	:USER-ASSIGNED-NAME	Pretty name for the HTML user interface
slotType	:DIRECT-TYPE	Meta class of the slot
protegeValueType	:SLOT-VALUE-TYPE	Value type from Protégé
allowedCls	:SLOT-VALUE-TYPE	Allowed Cls for Instance types.
defaultValues	:SLOT-DEFAULTS	Default value for the slot
slotInverse	:SLOT-INVERSE	Inverse of the slot
numericMin	:SLOT-NUMERIC-MINIMUM	Lower bound of a numeric data element
numericMax	:SLOT-NUMERIC-MAXIMUM	Upper bound of a numeric data element
cardinalityMin	:SLOT-MINIMUM-CARDINALITY	Minimum allowed data
cardinalityMax	:SLOT-MAXIMUM-CARDINALITY	Maximum allowed data
nullable		A flag for :SLOT-MINIMUM-CARDINALITY >== 1
isMultiple		A flag for :SLOT-MAXIMUM-CARDINALITY = -1
unique	:UNIQUE	
index	:DATABASE-INDEX	(Not implemented fully)
symbolChoices	:SLOT-VALUE-TYPE	Allowed value set for simple types such as strings, numbers, integers.
unit	:UNIT	Unit for numeric attributes, e.g. km, pound, cm.
documentation	:DOCUMENTATION	Description of the slot
rdbAttributeName		Attribute name implemented in the database. Maybe auto-edited in the transformation program.
rdbTarget		Description of what the slot maps to. It can be a slot described as :RDB_ATTRIBUTE(*[slot name]*), or another class :RDB_CLASS(*[class name]*).
dbValueType	:DATABASE-TYPE	RDB value type either as specific in :DATABASE-TYPE or by default transformation rules (see Figure 5.11).
dbValueLength	:DATABASE-TYPE-PARAMETER	Length of Varchar type, either as specified in :DATABASE-TYPE-PARAMETER or by transformation rules (Chapter 5)
isAssociated		A flag for whether the attribute is implemented as being associated to the corresponding domainCls table.
expression	:VALUE-EXPRESSION	(Not implemented)
viewSequence	:VIEW-SEQUENCE	Appearing sequence in HTML form
formWidget	:FORM-WIDGET	Widget used for a HTML form for data editing.
formWidgetParam	:FORM-WIDGET-PARAMETER	Parameter for the HTML widget
viewWidget	:VIEW-WIDGET	Widget used for viewing the element in HTML
viewWidgetParam	:VIEW-WIDGET-PARAMETER	Parameter for viewing widget

Figure 4.8. Listing of the attributes in the meta-data table *:RDB_SLOT.* The attributes are mostly mapped to (implemented) facets listed in Figure 4.5. Comments are available to ones that do not have a direct one-to-one match.

4.4.2. Meta-data storage

Meta-data about the mappings between Protégé and this relational schema are stored using the same schema in each Seedpod database instance. The schema for these meta-data tables are non-model-specific. In other words, they have the same schema regardless of which database they reside in. Their content is pre-populated by the transformation program.

There are two meta-data tables in the database: one for Protégé classes called *:RDB_CLASS* (Figure 4.7) *and the other for* slots called *:RDB_SLOT* (Figure 4.8). The Protégé facets for class *:RDB_SLOT* listed in Figure 4.5 are mapped to attributes for the table in Figure 4.8. The same mapping is true for attributes of *:RDB_CLASS.*

The two meta-data tables serialize the transformation, mappings between Protégé and the relational model. It allows the Seedpod application to query about the Protégé model while keeping in touch with the database implementation. It contains information about the object structure, how objects are stored in the relational database, and finally display customization for HTML pages. Examples of the meta-data tables can be found in Chapter 5.

Additionally, the meta-data tables divorce the dependency of the Seedpod application from Protégé. In the case that a more appropriate modeling environment is designed to replace Protégé, Seedpod's web application (Wikipedia: Web application, 2009) can still work as long as it produces meta-data tables. When changes need to be made to the model, a database engineer would need to translate the changes to meta-data table changes in the database. The web application is not affected. See Section 6.2.5 for more discussion on system evolution.

4.5. Web Server Application

Once a Seedpod Protégé model is transformed to a relational database schema, a Seedpod LIMS web application can be configured and installed to run with no programming involved. This section describes Seedpod's web server application, which dynamically generates web-based applications that allows users to manage data in the database (see component 3 in Figure 4.1). The focus of the description is in the technical implementation. One of the most salient characteristics of this web server application is the fact that it is non-domain specific. This means the application code does not contain any specific information about a particular experiment, scientist, or laboratory. Note that the implementation differs from auto-code generation in which partial API code is generated requiring manual completion such as Fogh's work in 2005 (Fogh, et al., 2005).

The web server is developed using JAVA Enterprise Edition (Wikipedia: Java Platform Enterprise Edition). A mix of JAVA server pages (JSP), JAVA Servlets, and JAVA classes can be found in the code base. The server application code is organized in general into model, view, and controller. The JAVA package *seedpod.webapp* contains view and controller components. It shares package *seedpod.model* with *seedpod.kb2db* (transformation). The server application runs on a Tomcat web server and it communicates with a PostgreSQL database.

4.5.1. "Model"

The section title "Model" may lead to confusion. This section is about the object abstraction for the application in JAVA code. Model here is an abstraction of a LIMS model, or meta-model, which makes a Protégé LIMS model an instance of the application model. To illustrate it with an example, a typical LIMS program may have an object model that includes object classes such as protocol, experiment, patient subject, etc. However, in Seedpod, the model classes consist of meta-classes such *RDBCls* and *RDBSlot* from the Protégé model and *Relation* and *Attribute* from the RDB model. This abstraction disregards the actual data types and allows the application to be general. The server application is hence not domain application specific.

A class called *ModelMap* captures mappings between Protégé and RDB. This is the heart of Seedpod, which is shared between the web application and the transformation program. The transformation program uses *ModelMap* to materialize the mapping into meta-data database tables (see Chapter 5). The web application imports the *ModelMap* object on startup from the database into a set of *ClsMap* (Protégé class) and *SlotMap* (Protégé slot) objects. These objects are similar to *RdbCls* and *RdbSlot* used for transformation. The distinction is that *ClsMap* and *SlotMap* are derived from the database serialization and they no longer have references to the original Protégé *Cls* and *Slot* objects like *RdbCls* and *RdbSlot.* The *ModelMap* object informs the behavior of the controller and view components of the web application as described in the next two sections.

Seedpod implements a universal unique object system. Instead of having each table manage its own unique primary key, the entire database manages one set of unique IDs through one table called *Thing.* Every data object instance added to the database is first added to *Thing* to obtain a new ID. That ID is then used to insert the object into its appropriate data table. This index allows the application to figure out quickly the data type of an instance by querying only one table. Additionally, the *Thing* table keeps track of the state of an object, whether it is saved or deleted. A Seedpod object instance is never deleted from the database.

In general, Seedpod treats all instances in the database as Seedpod data objects, or *SeedpodDO.* It implements *PersistenceDO,* which provides a "*CRUD*" interface for Creating, Retrieving, Updating, and Deleting of an object from the database. Each *SeedpodDO* manages a set of *AVPair* which stands for attribute-value pair. It is responsible for binding values to attributes. An *AVPair* object implements a set value and a get value function, validates data value(s), and generates a unique reference ID used by the user interface. Class *Relationship* captures associations between *SeedpodDO* objects. It has references to the relationship *SlotMap, and* source and target *SeedpodDO.*

4.5.2. Controller

The controller is also the logic of the application. It receives inputs, calls upon the model, and generates views. It is the interface between the view and the model of an application. Context variables of the LIMS are defined in a configuration file *web.xml* and accessible through class *LIMSContext* (see 4.7.2).

When the server is started, class *Seedpod* is invoked by the server which initiates a connection pool to the PostgreSQL database, and downloads the *ModelMap* from the metadata tables. The entire *ModelMap* is kept in memory for access for the life of the server application.

The connection pool allows a maximum number of 50 threads to be connected to the database at a time. A particular query may request an available or free connection from the pool. If no available thread exists, a new one is created.

Each time a page is requested, user authentication is validated by a filter JAVA servlet. If the user is authenticated, she is then led to the requested page. If not, she is then redirected to the login page. User authentication is saved for a browser session and is lost when the browser is closed. User passwords are encrypted using a *BASE64Encoder* hash function before saving to the database.

Seedpod also implements a persistence manager, *PManager.* This manager keeps a reference to a database connection, and sends queries to the RDB to retrieve data by Seedpod data objects. *PManager* creates *SeedpodDO* by object ID and/or object type. In fact, *PManager.getObject()* is designed to retrieve implemented objects by name using JAVA reflection to allow the application to be flexible (see 4.6.2 for more detail).

In addition to managing application communication with the database and authentication logic, the controller package includes major roles in accepting user requests from the browser, mostly processing HTML form submissions. These are classes found in the code base *seedpod.webapp.controller* package with names starting with *Action* such as *ActionInplaceEditor, ActionNewInstance,* etc. These classes are named because they are values to the action attribute in HTML form elements. They extend class *HttpServlet* and override functions *doPost()* and *doGet().* For example, the function

ActionNewInstance.doPost() asks *ModelMap* for the slots and their form element reference IDs (created by each slot's *AVPair* object). Then it retrieves the user submitted values by those reference IDs. The values are validated by each corresponding form widget. Finally, if no validation error is generated, an object is created in the database. The user is redirected to view the new object page. If a validation error occurs, the user is redirected back to the HTML form with error message prompts.

4.5.3. View

Most of the view pages are implemented in JSP pages. For example, *instance.jsp* provides layout of the html page. It calls a JAVA class *InstanceRenderer* passing a *SeedpodDO* object for the actual rendering. The *InstanceRenderer* has access to the object's meta-data through *ClsMap* and *SlotMaps.* It also has functions for rendering the *SeedpodDO* based on user request, whether it's for viewing, creating new or editing. For example, function *renderCreateForm()* creates an HTML form for users to input data for a new object. The function iterates through the *SeedpodDO*'s attribute-value pairs (*AVPair*). For each *AVPair*, a corresponding form widget is retrieved from the *LimsWidgetFactory* by name and then rendered. Again, each *AVPair* generates a reference ID for the form element which is used by the form handling class to retrieve the value of user input as described in 4.5.2.

A user can specify a HTML form widget and a view widget in the Protégé model. If they are not specified, a default widget based on Protégé data type is assigned during the transformation step. Each of the widgets is associated with an actual JAVA class that implements it for either viewing or editing. Figure 4.9 lists the available widgets, valid

Protégé and RDB types, and their corresponding JAVA classes which implement the functions. The JAVA classes extend (or inherit) a generic widget *LimsWidget* and override functions for rendering. Each widget class overrides a validation function to make sure that

Form Widget	**View Widget**	**Protégé type**	**RDB type**	**JAVA class name**	**Description**
TEXT	STRING	Integer, String	varchar(n), text	TextArea	HTML input that allows a user to input a string. The string length can be restricted based on the length for varchar.
RADIO	RADIO	Boolean	boolean	Radio	Allows users to select between allowed-value options.
SELECT		Symbol	varchar	Select	Shows a drop down boxes of allowed-value options.
DATE	DATE	String	varchar, text	Date	A calendar window pops up for user to choose a date which auto-fills the text string.
TEXTAREA		String	text	TextArea	A multi-row text input widget.
CHECKBOX	CHECKBOX	Symbol	varchar(n)	CheckBox	A check box widget similar to radio but allows multiple options being selected.
NUMERIC	NUMERIC	Float	numeric	Numeric	A numeric input that has pre-defined unit.
OBJECT_LINK	OBJECT_LINK	Instance	relation	Object_Link	A widget that allows the user to create relationships between two different object instances. See Figure 4.14 for example.
PASSWORD	PASSWORD	String	varchar	Password	A password string input that shows a star for each character of the password (Figure 4.12).
FILE-RESOURCE	FILE-RESOURCE	Instance of File	relation	File	A special OBJECT_LINK for object File instances.
	SPREADSHEET	Instance		SpreadSheet	Shows a tabular view of a set of instances.

Figure 4.9. This is a list of implemented HTML widgets. Form widgets are used in data input forms while view widgets are elements in rendering the data. Each widget has a valid data type it can work with. Each widget can do either or both view and edit. The widget names are parts of the Protégé model. The widgets instances are dynamically instantiated at run time by *LimsWidgetFactory.*

the data input or data it is asked to render can be handled by the rendering function. For example, this gives the widget a chance to display error messages for inappropriate user input. These widgets are organized by *LimsWidgetFactory.* The factory class dynamically initializes and instantiates these widgets by their names. Names of the widgets are part of the Protégé model.

Finally, page context sensitive content is implemented using AJAX, or JavaScript with XML (Wikipedia: Ajax). The content can be requested from the client to server asynchronously, increasing code modularity and interactivity. For example, in a page rendering a data object of type *Family*, a tool box on the right displays quick links for creating another new instance of *Family*, or browsing instances of *Family*, in addition to other unimplemented functions. A second content box below shows meta-information about the object, such as when it was created and by whom (this is not fully implemented but the information is available through a Seedpod system table *Access_Log*). The content of these boxes are updated depending on what the user is trying to do on a particular page. Independent AJAX functions call different JAVA Servlets to generate the content. An update only occurs to that portion of the page. See Figure 4.14 (right panel) for examples of these context sensitive AJAX boxes.

4.6. Extending and Customizing Seedpod

Seedpod server's application code is not domain-model specific. Unlike most of the other model-driven applications, Seedpod does not generate program code from the model. In the case that application code is generated, developers can go in to modify the generated code before deployment. In the case of Seedpod, adding features or making changes to existing features is not as trivial. The server application has designed hooks for developers to make extensions. This section describes three ways that one can extend and customize Seedpod: widgets, data objects, and HTML page display. The customization discussed here requires a knowledgeable programmer.

4.6.1. Customizable widgets

Seedpod's widgets associated with slots are customizable components. A widget is an HTML element. It has a unique ID. It may be a part of a form, in which case, it is editable. It can receive user data input (doPost), validates the input (validateSubmissionData), or render a form element (doEdit). Alternatively, it may just be used to render a data object (doView, or render). A developer can simply extend the generic *LimsWidget* class and override the following functions:

- *Constructor(AVPair avpair):* initializes a widget. The AVPair object generates the HTML element widget ID.
- *setId:* sets a string ID name for the HTML widget.
- *getId:* returns a string ID name for the HTML widget.
- *Protected String render:* This function is called to generate the HTML code which is called by either doEdit or doView depending on the widget's function.

- *validateWidget*: validates the AVPair value type against what the widget can handle.
- *String doEdit*: returns an HTML form element that the user can interact with.
- *String doView*: returns an HTML element that renders the data value.
- *String doPost(SeedpodDO obj, Object input)*: the input object is a value assigned to this widgets AVPair.
- *SlotMap getAttribute*: returns the attribute part of the associated AVPair object.
- *Object getData:* returns the value part of the associated AVPair object.
- *boolean allowInPlaceEdit*: returns true if the form widget responds to an AJAX call for real time edit of an element.
- *boolean validateSubmissionData (Object submittedValue)*: returns true if the submitted data is appropriate for input. For example, it may check that the data is not null for an attribute that requires an input.
- *boolean supportsMultiValueInput*: returns true if the widget can accept multiple value input, or render a set of values. The spreadsheet widget is an example of a widget that can support a set of values.

Then, to make the new widget available to be used in a model, the name of the widget is added to *SeedpodModel.Form.RdbCls.FormWidget* enum list or *SeedpodModel.Form.RdbCls.FormWidget* enum list. Alternatively, the name is added manually to the meta-model *:RDB_SLOT* as allowed values. The function *LimsWidgetFactory.getWidget()* is modified such that the switch block would return a new widget instance when the widget's name is requested.

4.6.2. Extensible object definitions

SeedpodDO is a generic database object that is persistent in the database. Customized persistence objects can extend *SeedpodDO* to have additional functionalities. As it is implemented, as long as the JAVA class has the same name as what is being modeled in Protégé in addition to have the class being placed inside of package *seedpod.model.custom, PManager* can find the class and create an instance of it by reflection. Class *SeedpodUser* is an example of a custom class. It implements function *authenticate* which encrypts a user input password and compares it with what is stored in the database.

4.6.3. Extensible page layout

In addition to new object definition, developers can also develop a new SeedpodDO *InstanceRenderer* instead of the default. This renderer can change the layout of an object display on an HTML page, or change the look and feel of a data input form. New JSP pages can be developed to augment what the user interface looks like as well. An important point to keep in mind is that changes to modeled classes may make these extension classes or JSP pages compromised.

4.7. Application Workflow

An important value of Seedpod is the ease of deployment. The entire project is built using open source technology. This section describes the major steps in deploying a Seedpod application. Even though Seedpod was designed for scientists to launch a full relational-database-backed web application, at this point, an informaticist works with scientific researchers as a team in the process. The informaticist may not need to be a programmer or familiar with database. A minimal amount of knowledge in software installation and server administration is needed.

Seedpod web server application code is packaged along with the transformation code. The whole package is open source available for download from Google Code URL: http://code.google.com/p/seedpod/. It is released under GNU Public License V3.0 (Free Software Foundation, 2007).

4.7.1. Step 1: create the model

A scientist user is only involved in the first step of the development process. The user designs a LIMS model in Protégé. Protégé can be downloaded from Stanford's website. It is a platform independent application. Seedpod's meta-model class and slot added by Seedpod's Protégé plug-in (Figure 4.4 and Figure 4.5) must be used. If the scientist researcher is not familiar with modeling in Protégé, an informaticist works with the scientist and interviews the laboratory researchers and technicians about data flow in the lab, experiment protocols, and other requirements.

4.7.2. Step 2: Transform model and create database

Once a Protégé model is complete, the informaticist can run the transformation application either by using the Protégé transform plug-in or by running the JAVA transform application on the command line. The resulting SQL files are saved. The next step is to install a relational database engine such as PostgreSQL used in this example. The SQL files created by the transformation are run in the database engine to create a new database. Finally, the database server is started. The database server connection URL is saved for configuring the webapp in the next step.

4.7.3. Step 3: Deploy web application

In the last step, an Apache Tomcat web server (The Apache Software Foundation, 2011) is installed and the webapp is downloaded. The file *web.xml* in the web application must be configured. Figure 4.10 lists the parameters that need to be set correctly for the webapp to talk to the database server. The HTML pages and compiled java classes should reside in the WebContent folder. The project is compiled into a Web Application Archive, or WAR, file for deployment. The Apache Tomcat server is started and the WAR file is deployed. The webapp can be accessed from a browser using URL: *[web server domain]/[Seedpod app install path]/lims/index.jsp.* Upon the first execution of the webapp, a default user *Administrator* with password *seedpod* is automatically created in the database. A system administrator can login with the default account and change the password in the user configuration page.

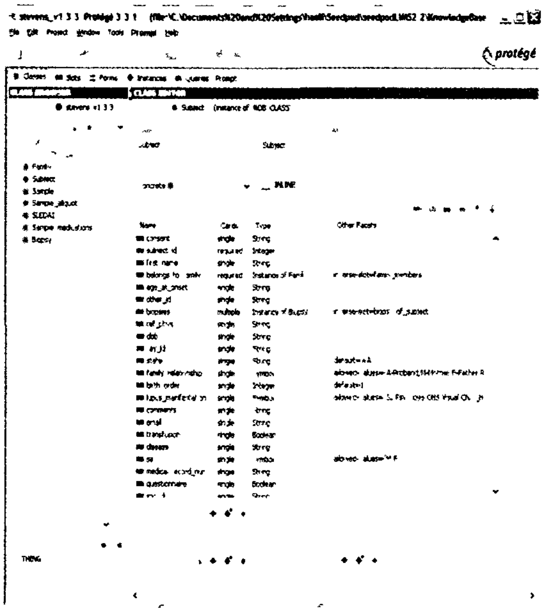
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
      xmlns="http://java.sun.com/xml/ns/j2ee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

      ...
      <context-param>
            <param-name>applicationDirectory</param-name>
            <param-value>/lims</param-value>
      </context-param>

      <context-param>
            <param-name>WebappURL</param-name>
      <param-value>http://localhost:8088/</param-value>
      </context-param>

      <context-param>
            <param-name>LIMSName</param-name>
            <param-value>Steven's Lupus Lap</param-value>
      </context-param>

      <context-param>
            <param-name>DatabaseDriver</param-name>
            <param-value>org.postgresql.Driver</param-value>
      </context-param>

      <context-param>
            <param-name>DatabaseURL</param-name>
            <param-value>
      jdbc:postgresql://localhost:5432/stevens_v1.3/
            </param-value>
      </context-param>

      <context-param>
            <param-name>DatabaseUser</param-name>
            <param-value>postgres</param-value>
      </context-param>

      <context-param>
            <param-name>DatabasePassword</param-name>
            <param-value>postgres</param-value>
      </context-param>
      <context-param>
            <param-name>DatabaseName</param-name>
            <param-value>Stevens(v1.3)</param-value>
      </context-param>
...
</webapp>
```

Figure 4.10. Here is a snippet of web.xml on the web server which is required to be configured for the application to communicate with the database server.

This last step would allow scientists to come back to test the system before it gets final deployment. A scientist user may need to tweak the model in step 1. An informaticist may need to debug or customize the web-based GUI in step 2. Short cycles of testing and deployment may occur before the LIMS is ready to store real data.

4.8. Results

Steven's Lupus Research Lab (2.1.2) was studied for a demonstration of Seedpod. The author interviewed scientists that work in the lab for 2-3 hours and completed a Protégé model in 2 hours. In general, the process for an informaticist to understand the LIMS model is the majority of the effort. This section shows screenshots from the Steven's Protégé model and the web-based user interface.


stevens_v1 3 3 Protégé 3 3 1 (file:\C:\Documents%20and%20Settings\...\Seedpod\seedpod.LIMS2 2\KnowledgeBase
File Edit Project Window Tools Prompt Help
protégé
Classes Slots Forms Instances Queries Prompt
CLASS BROWSER
CLASS EDITOR
stevens v1 3 3
Subject (instance of :NOB CLASS
Subject
Subject
Concrete
INLINE
Family
Subject
Sample
Sample_aliquot
SLEDAI
Sample medications
Biopsy
THING

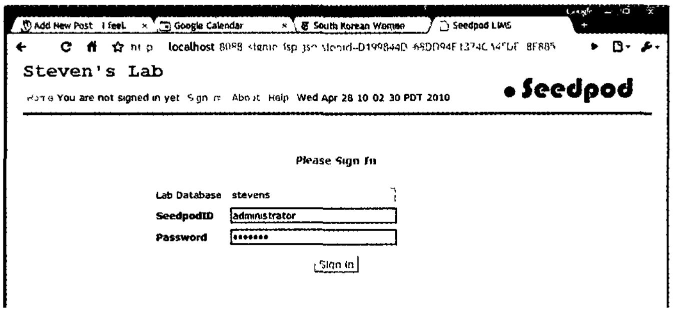
Name	Cardi.	Type	Other Facets
consent	single	String	
subject_id	required	Integer	
first_name	single	String	
belongs_to_family	required	Instance of Family	inverse-slot=family_members
age_at_onset	single	String	
other_id	single	String	
biopsies	multiple	Instance of Biopsy	inverse-slot=biopsy_of_subject
ref_phys	single	String	
dob	single	String	
ary_id	single	String	
state	single	String	default=WA
family_relationship	single	Symbol	allowed-values=A-Proband,M-Mother,F-Father,R
birth_order	single	Integer	default=1
lupus_manifestation	single	Symbol	allowed-values=S,Pin,...CNS,Visual,Ch
comments	single	String	
email	single	String	
transfusion	single	Boolean	
disease	single	String	
sex	single	Symbol	allowed-values=M,F
medical_record_num	single	String	
questionnaire	single	Boolean	

Figure 4.11. A screen shot of the Lupus Lab model.

4.8.1. Steven's Lab Protégé Model

Figure 4.11 shows a screen shot of the Protégé model. Class *Subject* is highlighted. The template slots panel on the right shows some simple data types and some relationships. For example, *belongs_to_family* is a reference to a *Family* object.

4.8.2. Web-based User Interface



Steven's Lab

You are not signed in yet   Wed Apr 28 10 02 30 PDT 2010

Please Sign In

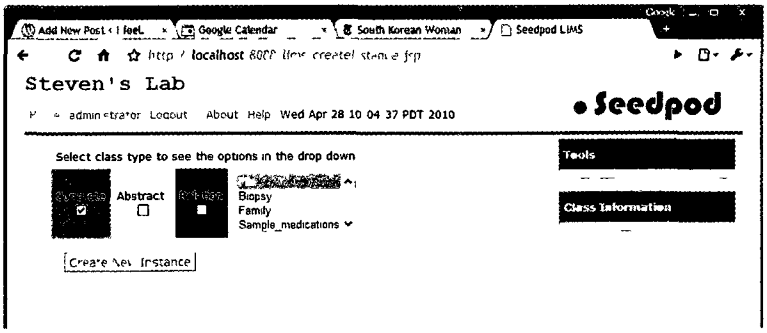
Lab Database stevens

SeedpodID administrator

Password •••••••

Figure 4.12. User log in screen.

This section shows the web screenshots of the user interface for various data management tasks. Figure 4.12 shows a screen for user sign in. Figure 4.13 shows the screen for a user to choose the class of which she wishes to create a new instance. The dropdown list of classes is dynamically generated from the model.



Steven's Lab

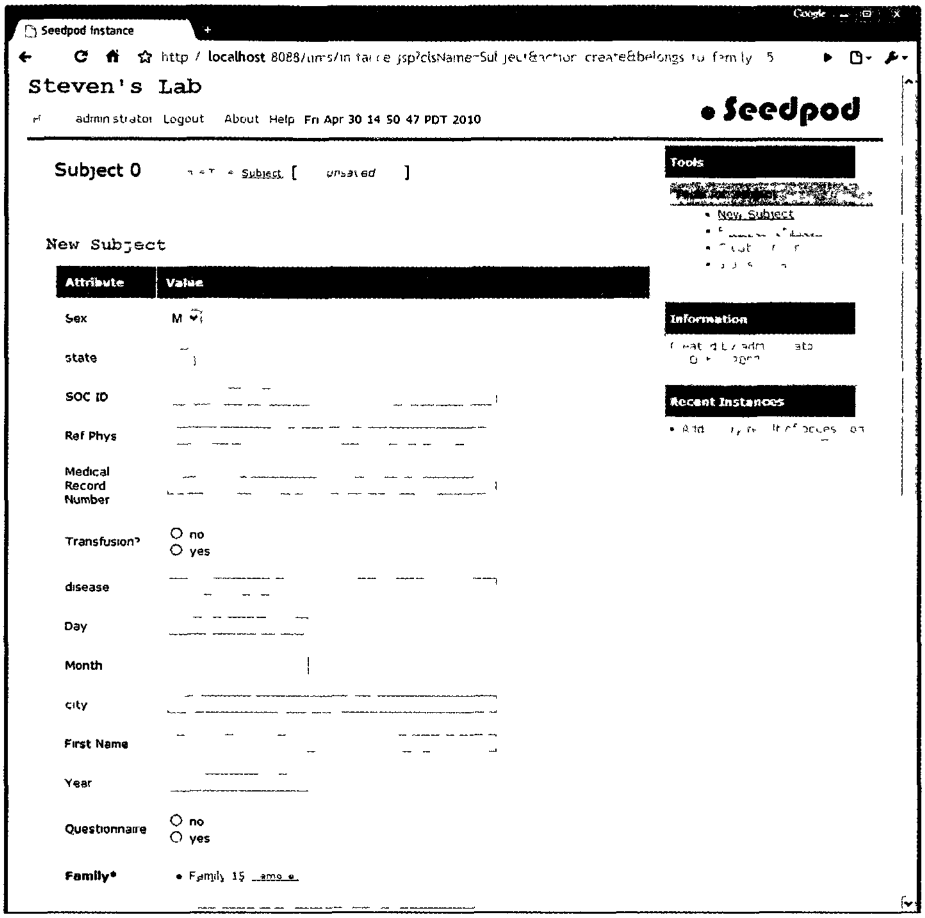
Wed Apr 28 10 04 37 PDT 2010

Select class type to see the options in the drop down

☑ Abstract □ ■

Biopsy
Family
Sample_medications

Figure 4.13. Choose a class type for creating a new instance.



Seedpod instance

http / localhost 8088/ums/in tance jsp?clsName=Subject&action create&belongs to family 5

Steven's Lab

administrator Logout About Help Fri Apr 30 14 50 47 PDT 2010

• Seedpod

Subject 0 Subject [unsaved]

New Subject

Attribute	Value
Sex	M
state	
SOC ID	
Ref Phys	
Medical Record Number	
Transfusion?	○ no ○ yes
disease	
Day	
Month	
city	
First Name	
Year	
Questionnaire	○ no ○ yes
Family*	• Family 15 remove

Tools

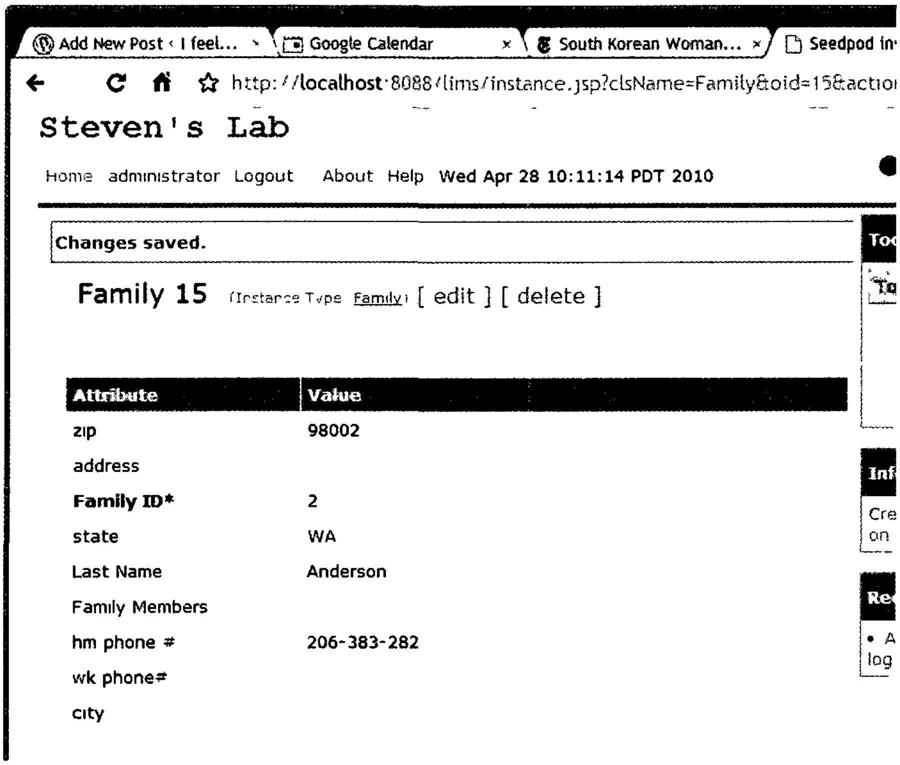
- New Subject

Information

Recent Instances

Figure 4.14. The web form for creating a new instance of *Subject.*

Figure 4.14 shows the web form for creating a new instance, in this case a *Subject.* The attributes and corresponding widgets are dynamically generated. A bolded attribute name denotes that a value is required. The names for attributes are user-defined displayed names, which are different from the names used in the model or attribute names in the database tables.


Add New Post ‹ I feel…
Google Calendar
South Korean Woman…
Seedpod In
http://localhost:8088/lims/instance.jsp?clsName=Family&oid=15&actio
Home administrator Logout About Help
To
Inf
Cre
on
Re
• A
log

Steven's Lab

Wed Apr 28 10:11:14 PDT 2010

Changes saved.

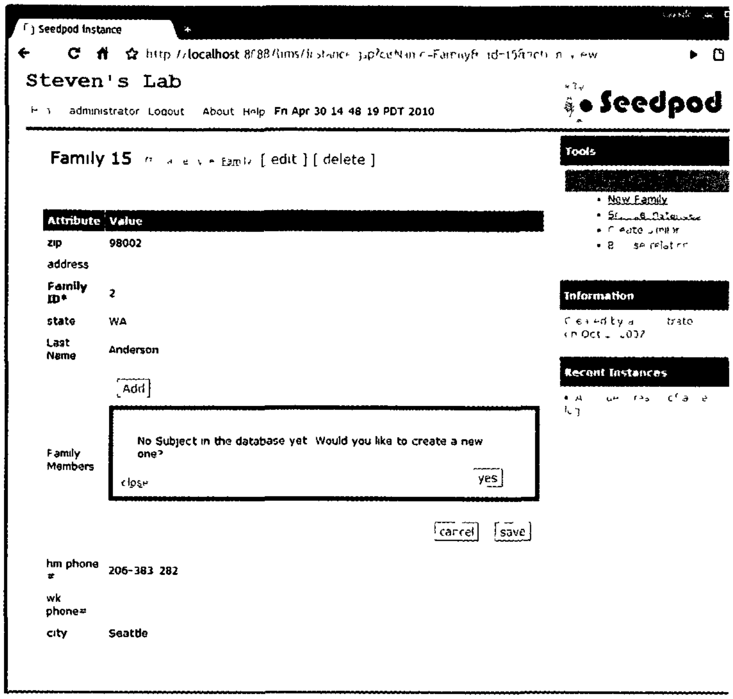
Family 15 (Instance Type Family) [edit] [delete]

Attribute	Value
zip	98002
address	
Family ID*	2
state	WA
Last Name	Anderson
Family Members	
hm phone #	206-383-282
wk phone#	
city	

Figure 4.15. An example of an instance view page.

Figure 4.15 shows an example of an instance view page after it has been edited. A message that says "Changes saved" is prompted at the top. Each attribute can be edited in place individually by clicking on the corresponding value cell on the right. Actions such as edit and delete are provided next to the object instance name.

Figure 4.16 shows that *Family Members* is a relationship attribute. Clicking on the value cell shows the user that no subject exists for this *Family* instance yet. The user can choose to add one from the database or create a new one. If the user chooses to create a new instance, she is then brought to a screen that is shown in Figure 4.14 with the value for *Family* automatically filled out. If the user chooses to add a relationship to an object that



Seedpod Instance

http://localhost 8788/lims/

Steven's Lab

administrator Logout About Help Fri Apr 30 14 48 19 PDT 2010

Seedpod

Family 15 [edit] [delete]

Attribute	Value
zip	98002
address	
Family ID*	2
state	WA
Last Name	Anderson
Family Members	Add No Subject in the database yet Would you like to create a new one? close yes
	cancel save
hm phone #	206-383 282
wk phone#	
city	Seattle

Tools
- New Family

Information

Oct

Recent Instances

Figure 4.16. Editing a relationship value shows options for choosing a new instance or creating a new instance. This is an example of the OBJECT_LINK widget.

already exists in the database, Figure 4.17 shows a listing of available *Family* instances that could be added as a value to this *Subject* instance's relationship to *Family.*

In summary, for the Stevens Lab, the author developed a Protégé model for the study. Seedpod automatically generated a relational database and a webapp. The webapp dynamically generates data entry and editing web forms, data browsing HTML pages in addition to user management and login pages.

4.9. Conclusion

This chapter describes the prototype application Seedpod developed to demonstrate the ideas behind the MDA LIMS of this thesis. Detailed documentation on the

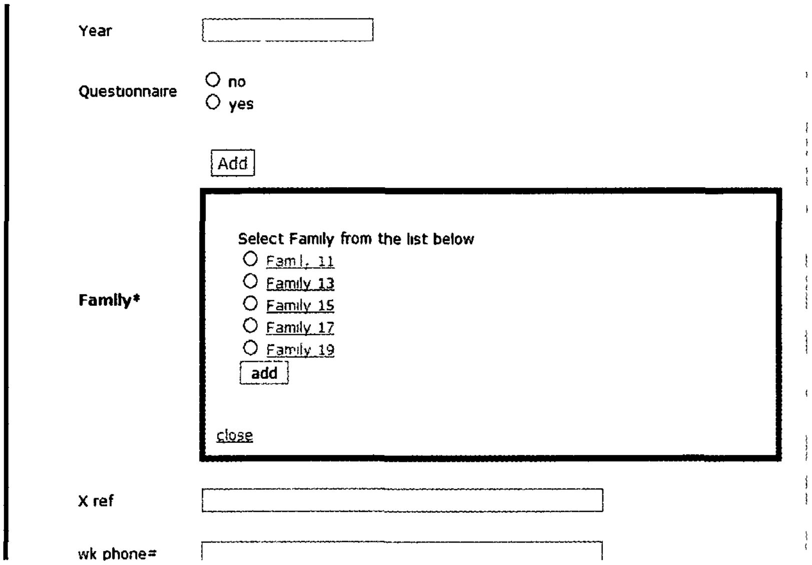


Figure 4.17. A listing of existing *Family* instances is shown as allowed values to be added for this *Subject* instance.

architectural design, implementation of the components, installation and deployment of Seedpod are included for the interest of informaticists. The Protégé model and samples resulting web-based GUI are shown for the perspective of scientific users. Chapter 5 delves into the transformation method while Chapter 6 analyzes how well Seedpod meets the LIMS requirements.

5. FRAME-BASED MODEL TO RELATIONAL MODEL TRANSFORMATION

Seedpod users can model their LIMS using Protégé's graphical user interface (GUI). Protégé models are frame-based, an object-oriented like representation language (Minsky, 1974). Seedpod stores experiment data in a normalized relational database. On the one hand, this setup allows Seedpod to take advantage of both Protégé's expressive modeling environment and the relational databases' (RDB) robust storage and retrieval capability. On the other hand, the modeling language uses a different approach and vocabulary than the storage language. One must translate the frame-based Protégé model to a relational model for the RDB. To avoid manual translation for individual Protégé models, a generic automated process called *model transformation* is necessary to increase efficiency and accuracy. This is an example of ModelGen, one of the model management operators described by Microsoft researcher Bernstein (Atzeni, Cappellari, & Bernstein, 2005; Bernstein, 2003).

This chapter describes this generalized transformation method for automatic translation from a Protégé model to a RDB schema. This work was conducted in collaboration with Dr. John Gennari and Dr. Peter Mork (Gennari, Mork, & Li, 2005). Before diving into the transformation method, the chapter defines key concepts used such as frame-based model, model and meta-model. The transformation method consists of a set of rules described in Section 5.2. Implementation details of the rules are in section 5.3 with

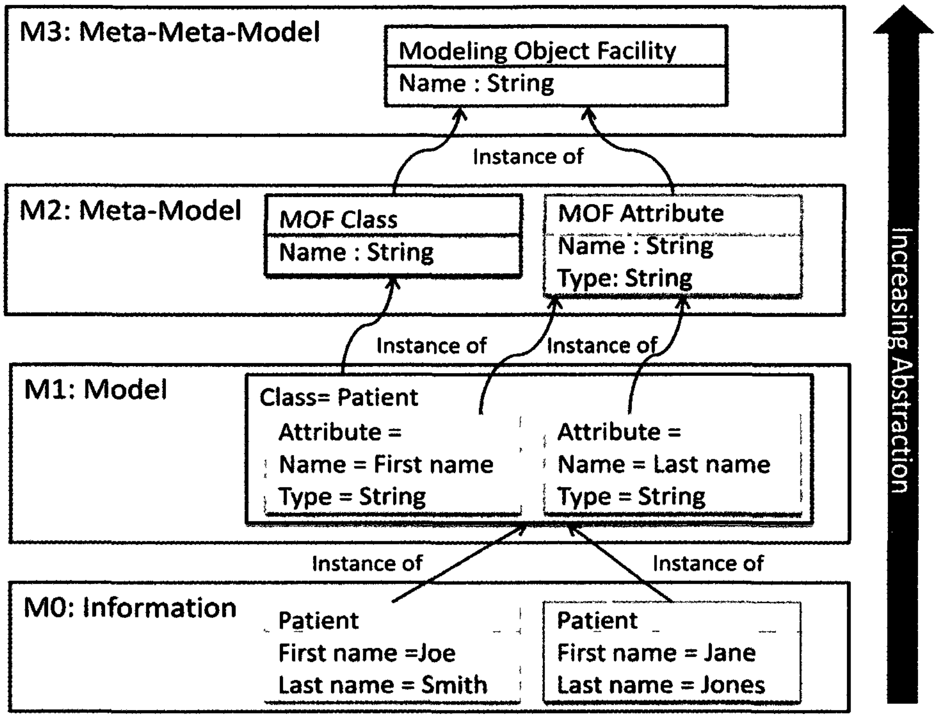


Figure 5.1. Four-layer model architecture shows a model is an instance of a meta-model. A frame-based model and a relational database schema are examples of models. Transformation rules are defined using constructs of meta-models.

additional built-in features specific for Seedpod. Section 5.4 shows results of the transformation comparing the Protégé input with the RDB output.

5.1. Meta-Model and Model Architecture

Automatic transformation relies on the standardized four-layer modeling architecture of Object Modeling Group (OMG). An understanding of the architecture allows one to see the abstraction relation between meta-models and models. So first, in this section, the model architecture, and definitions of frame-based models and relational models are defined.

5.1.1. Four modeling layers of OMG

The transformation method described in the next section is based on OMG's four-layer architecture, which includes meta-meta model, meta-model, model, and information (Figure 5.1). This OMG framework was shown effective in describing a complex information management system (Kleppe, Warmer, & Bast, 2003; MDA, 2010; Estrella F. , Kovacs, Goff, & McClatchey, 2001). The four layers are called M0, M1, M2, and M3.

M0 is the information layer, containing real instances. For example, a patient named *Joe Smith* was scheduled for brain surgery on the day of *November 21, 2005* performed by *Dr Cass.* The italicized items are the data stored in M0.

The M1 layer contains models, for example relational schema, which describes the information elements. Following the above example, in the M1 layer, *Patient* is defined with properties such as *First Name* and *Last Name.* The concepts of M1 are the classifications or definitions for instances in M0. M0 relates to M1 through an *is-an-instance-of* relationship.

M2 defines constructs used in M1. The concepts defined in M1 are instances of M2 layer constructs. *Patient* is an instance of the *Class* construct. *First name* and *Last name* are defined using the *Attribute construct* in M2. This layer is also called the meta-model layer. In other words, M2 provides the language one uses to construct the model in M1, e.g. *Class, Attribute, Relationship.*

M3 is called the meta-meta-model layer which defines the language used for specifying meta-models (Estrella F. , Kovacs, Goff, & McClatchey, 2001). A similar pattern of relationship between M0 and M1, M1 and M2 is observed between M2 and M3. Every element of M2 is an instance of an M3 element. For example, *Modeling Class* and *Modeling Attribute* are instances of Model Object Facility (MOF) (XMIBackendTechnicalBackground, 2006). MOF is the standard M3 layer defined by OMG.

In summary, a given layer is an instance of the layer above it, and the layer above is a conceptual abstraction, or meta-model of said layer. This four-layer model is transferable to both frame-based and relational models. Frame-based models and relational schemata are examples of M1 models for their respective information layers (M0): knowledge base and database.

Frame-based models and relational models have different design approaches. A relational model stresses explicit entity type constructors, while a frame-based model uses attributes to interrelate objects. The two may lead to fundamentally different schemata, or models (Hull & King, 1987). They do, however, share many similarities that facilitate automatic transformation. The following description of the two models provides definition of the terms used in the transformation rules, and helps to intuit the transformation rules in the next section.

5.1.2. Definition of a relational model

A relational model contains a set of named relations, or tables (Ramakrishnan & Gehrke, 2002). Each table contains a set of named attributes, or column headings. Each attribute has a defined primitive type. Primitive types include characters, text, integer, date, etc. Each attribute can only take on a single value for the type. There can be constraints on the tables and attributes such as cardinality, default values, and null-ability.

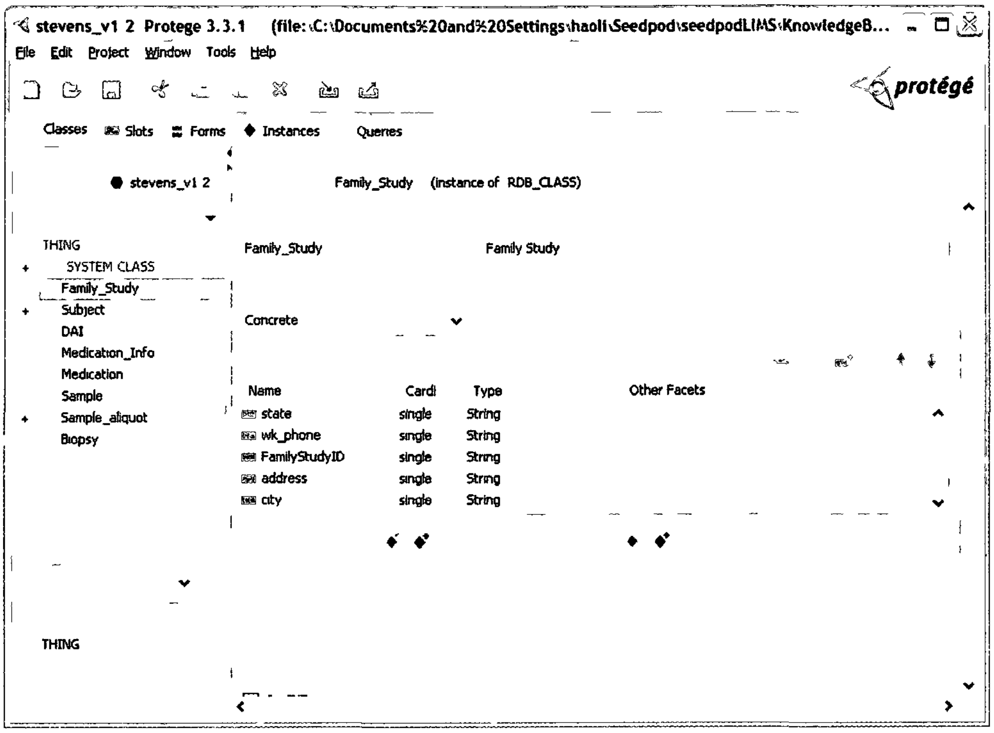
A relationship between entities, whether it is is-a, part-of, or association, can be implemented using foreign keys.

A relational model, or a relational schema, is defined using meta-model constructs: table, attribute, and foreign key. In this thesis, relational schema and relational model are used interchangeably. This definition of a relational model is simplified, devoid of concepts such as procedures, constraints, and indices in addition to vendor specific elements. What is of concern here is the concepts and their relationships.

5.1.3. Definition of a frame-based model

A frame-based model is similar to an object-oriented model. It consists of a set of classes, template slots, and facets (Gennari, et al., 2003). In other words, *class, slot,* and *facet* are part of the frame-based meta-model. A class contains a set of template slots. A slot is described by a set of facets. Classes are organized into a hierarchy in which template slots are passed on by a parent class to its descendants. Each class has a role, which declares the class to be either abstract or concrete. The distinction is that concrete classes can have direct instances whereas abstract classes cannot. Each template slot is a binary relation linking a class instance to a value. A value is constrained or defined by facets including type and cardinality restrictions. Values types can be string, integer, float, instance, class, any, or Boolean. A cardinality restriction may define the minimum participation requirement of a value to be 1. In other words, facets are properties of slots. Figure 5.2 shows an example of such a model in Protégé. More details about modeling in Protégé are found in Section 4.2.

A)


File Edit Project Window Tools Help
protégé
Classes Slots Forms Instances Queries

stevens_v1 2 Protege 3.3.1 (file:\C:\Documents%20and%20Settings\haoli\Seedpod\seedpodLIMS\KnowledgeB...

stevens_v1 2

THING
+ SYSTEM CLASS
Family_Study
+ Subject
DAI
Medication_Info
Medication
Sample
+ Sample_aliquot
Biopsy

THING

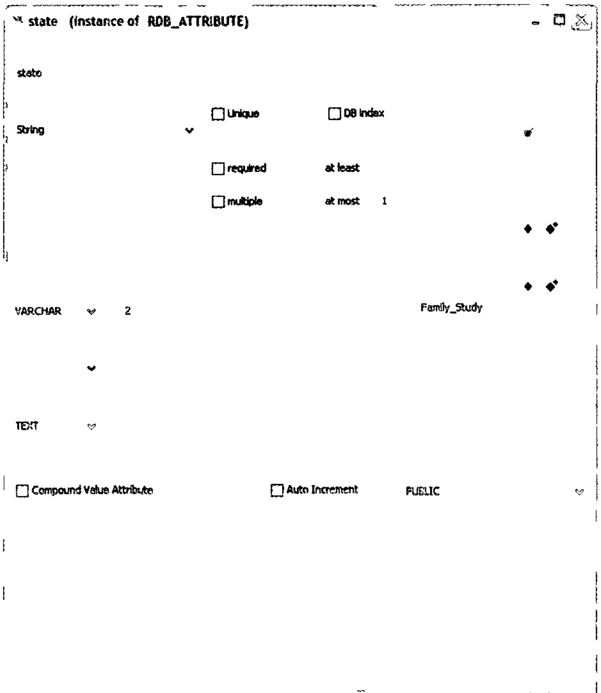
Family_Study (instance of RDB_CLASS)

Family_Study Family Study

Concrete

Name	Cardi	Type	Other Facets
state	single	String	
wk_phone	single	String	
FamilyStudyID	single	String	
address	single	String	
city	single	String	

B)



state (instance of RDB_ATTRIBUTE)

state

String

□ Unique □ DB Index

□ required at least

□ multiple at most 1

VARCHAR 2 Family_Study

TEXT

□ Compound Value Attribute □ Auto Increment PUBLIC

A)

Figure 5.2. A screenshot from Protégé. **A)** shows on the left hand side a class hierarchy, and the right side definition of a highlighted class. **B)** is a screen shot of the slot definition pane from Protégé.

5.2. Transformation Rules

The transformation method consists of a set of rules that map entities from a frame-based model to a relational model (Figure 5.3). Changes in the information layer require the models to change. However, the meta-models can remain stable; the constructs used to define the models (class, table, slot, attribute, etc) do not change. Automated transformation between two models is possible because the rules are defined using terms

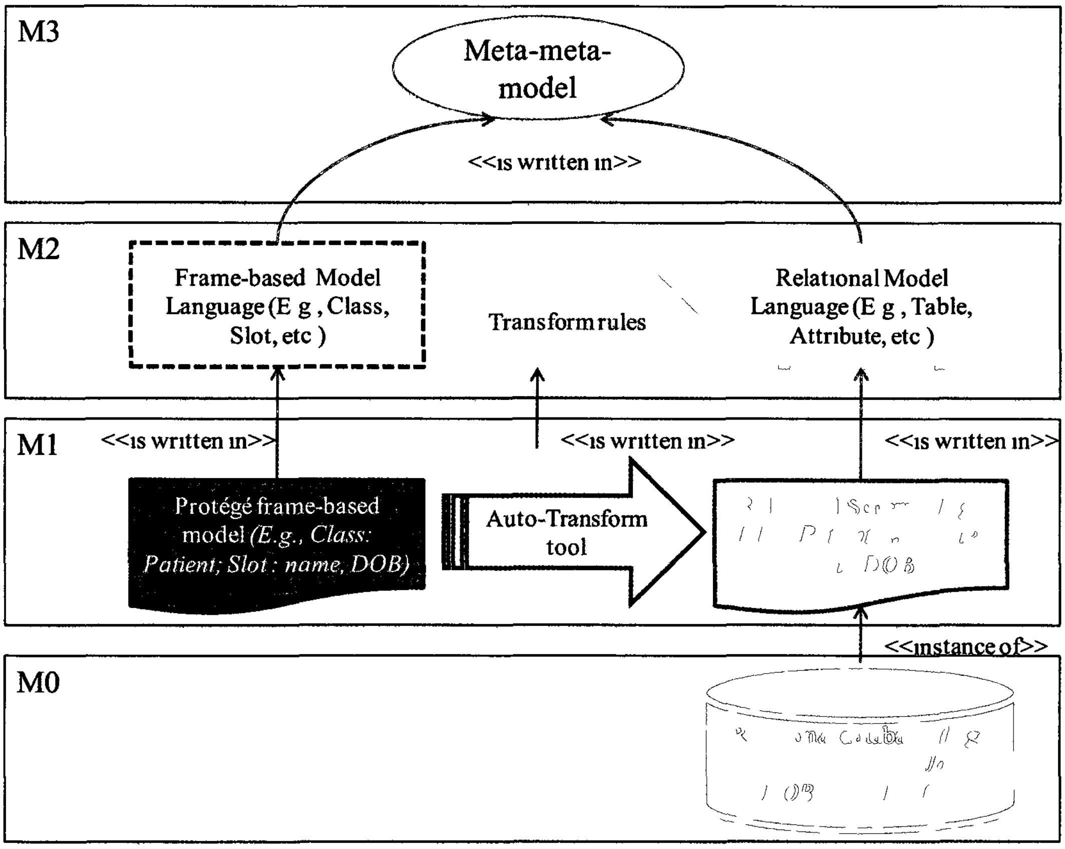

M3
Meta-meta-model
<<ıs wrıtten ın>>
M2
Frame-based Model Language (E g , Class, Slot, etc)
Transform rules
Relatıonal Model Language (E g , Table, Attrıbute, etc)
M1
<<ıs wrıtten ın>>
<<ıs wrıtten ın>>
<<ıs wrıtten ın>>
Protégé frame-based model (E.g., Class: Patient; Slot: name, DOB)
Auto-Transform tool
<<ınstance of>>
M0

Figure 5.3. Transformation of M0, M1, and M2. The M0 level is not transformed. Seedpod keeps data in the relational database but not in the Protégé side. Automatic transformation haopens at the M1 level written with constructs defined in M2.

from the meta-models, and hence the implementation of the transformation requires no details of actual models. As a result, there is no need to create ad hoc model-dependent, one-to-one mappings.

Intuitively, the transformation method is similar to the object-oriented model to relational model transformation (Niyomthum & Chittayasothorn, 2003). A class in a frame-based model becomes a table in a relational model. A slot becomes an attribute. However, the expressivity of a frame-based system necessitates a more complex set of rules than object-to-relational transformations. Impedance between object-oriented to the relational model transformation must also be dealt with here (Ambler, 2000). The set of transformation rules are as follows:

T1. *A class* ***C*** *is transformed to either a relational table or view* $\boldsymbol{R_C}$*, depending on the class's role descriptor and position in a hierarchy.*

a. If ***C*** *is a concrete class, then create a table* $\boldsymbol{R_C}$*, and add a primary key named* ***ID****.*

b. If ***C*** *is a non-leaf class, regardless of whether it is concrete or abstract, transform* ***C*** *to a table,* $\boldsymbol{R_C}$***. Then create a view,* $\boldsymbol{V_C}$*, which is the union of table* $\boldsymbol{R_C}$*** *and all the corresponding tables of* ***C****'s subclasses.*

T2. *A slot* ***S*** *of a class* ***C*** *is transformed depending on the slot's value type and cardinality.* ***S*** *can be either inherited from* ***C****'s super-class or owned by* ***C****.*

a. If ***S*** *has a primitive value type, i.e. String, Integer, Float, Boolean, and Symbol and has cardinality of 0 or 1, create an attribute,* $\boldsymbol{A_s}$*.* $\boldsymbol{A_s}$ *is* *associated with $\boldsymbol{R_C}$ (T1), and has a corresponding relational model primitive type.*

b. *If **S** is a type instance of a class **C'**, and has cardinality of 0 or 1, create a foreign key in table $\boldsymbol{R_C}$ named $\boldsymbol{F_C}$, which references the primary key in table $\boldsymbol{R_{C'}}$.*

c. *If **S** has cardinality multiple (regardless of its type being a primitive or an instance), create an association table, $\boldsymbol{R_S}$. Add foreign key $\boldsymbol{F_C}$ in $\boldsymbol{R_S}$ referencing $\boldsymbol{R_C}$'s primary key. Also in this association table $\boldsymbol{R_S}$, create **A** (Attribute(s)) for **S** according to single cardinality rules in **T2a** or **T2b**.*

There are two necessary assumptions about the frame-based model. The first assumption is that only the default standard meta-model is used. This is an important assumption because the meta-model is extensible to accommodate user defined meta-classes. During implementation, non-standard meta-model concepts would not be handled correctly or not at all.

The second assumption of the transformation method is that every slot is associated with a class. Slots are first-class objects; users can define slots without association with any classes. This method necessarily limits the transformation to only slots associated with classes, because they are the only ones that make sense in the relational model.

The set of rules defined in this section is generic. In practice, additional rules were also established when dealing with Protégé's frame-based model, especially one that has been customized for Seedpod. Implementation of the generic and additional rules is described in detail with implementation examples in the next section.

5.3. Implementation Details

In Seedpod, the frame-based model is designed in Protégé. The transformation is implemented in a JAVA program called *kb2db* packaged inside of Seedpod's application code. A Protégé project is input to the program, accessed using the programming API provided by Stanford. The output is a relational model written as a set of data definition language (DDL) statements in the form of SQL. The SQL statements conform to the SQL-99 standard (SQL:1999, 2011), which can be executed directly in relational database management systems (RDMS) that conform to the standard such as PostgreSQL. In addition to the relational model, the transformation also exports two meta-data tables that serialize the mapping data between the input Protégé project and the output RDB. This section describes the data structure, algorithm, and execution of the program.

5.3.1. Data structure

Since the transformation rules are defined in terms of meta-model concepts (e.g. Class and Slot for frame-based models, relation and attribute for RDB), the Java code only deals with these concepts. One would not find any specific model instance concept (e.g. experiment, patient, DOB). The meta-model resides in the *SYSTEM_CLS* hierarchy of Protégé. However, the transformation rules in 5.2 are generic with the assumption that the basic standard meta-slot class definition is used. As described in Chapter 4, Seedpod

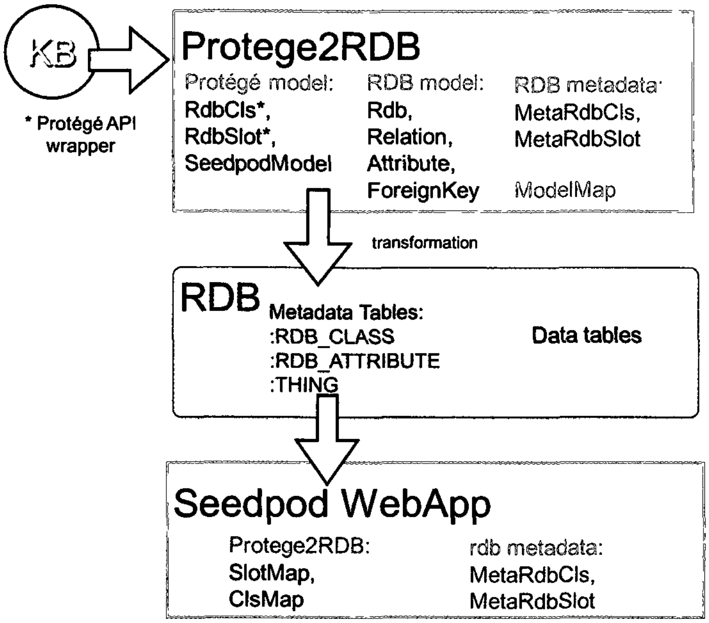


Figure 5.4. Organization of data objects in the transformation JAVA implementation.

expands on the standard meta-class and meta-slot concepts to their respective subclasses *:RDB_CLS* and *:RDB_SLOT* by adding additional properties. Therefore, in the implementation of the transformation, these additional facets are handled in new Java classes *RdbCls* and *RdbSlot*, which are, respectively, wrapper classes of Protégé API's *Cls* and *Slot*. Allowed values and names of these expanded facets are stored in the *SeedpodModel* class. For example, *:DATABASE-TYPE* is added to the Seedpod model's slot as a facet so that users can explicitly define database value types to avoid ambiguity (see Section 4.3). The data objects described in this section are also illustrated in Figure 5.4.

Similar to the Protégé model, there is a collection of Java classes that represent the RDB model such as *RDB, Relation, Attribute, ForeignKey, etc.* It may seem much more straightforward for the transform to read a Protégé file while writing out a SQL output. It turned out to be necessary to have an object representation of the RDB concepts for two reasons. One reason is that developers can customize or change the serialization of the RDB model by implementing additional exporters (examples described later).

The second reason for having an RDB model in the code is that the objects can be mapped to Protégé objects via an object *ModelMap*. *ModelMap* stores these mappings and serializes them in SQL using classes *MetaRdbCls* and *MetaRdbSlot*. Classes *MetaRdbCls* and *MetaRdbSlot* store schema of the meta-data schema defined in Section 5.5.1. They provide an interface between *ModelMap* and the actual storage in the database. *ModelMap* is also a wrapper object for *RdbCls* and *RdbSlot*, which are meta-data object classes used by the Seedpod webapp at runtime. *ModelMap* is exported as two meta-data tables as a result of the transformation, even though it is not part of the transformation algorithm.

5.3.2. Algorithm and implementation details

Before the transformation is run on a Protégé project, one can validate the model by calling function *validateKB()*, which uses *ProjectTransformValidator*. The validator reinforces the following three rules:

- The Protégé project must be a Clips project, i.e. not OWL or RDF, etc.
- The project uses the default Seedpod meta-class *:RDB_CLS* for its classes.
- The project uses the default Seedpod meta-class *:RDB_SLOT* for its slots.

The transformation rules are implemented in the *Protege2RDB.transform()* function (Figure 5.5) with the following algorithm.

Step 0. Initialize: Initializing data structures. Protégé's *Cls* is converted to *RdbCls*, *Slot* is converted to *RdbSlot*.

Step 1. Map inverse slots: Protégé models allow expression of inverse slots. For example, a class named *Study has* a slot *hasSubjects*, which specifies a value collection of instances of class *Subject*. Class *Subject* may then have a slot *belongToStudy* which specifies a value of Instance type class *Study*. Slots *hasSubjects* and *belongToStudy* may then have a defined reciprocal relationship defined using inverse slots in the Protégé

```
/**
 * Transform Protégé (Cls, Slot, Facet)
 * to RDB (Relation, Attribute, Foreign Key)
 */
public void transform() {
        // Step 0. initialize
        init();

        // Step 1. Hide one of the inverse slot pairs
        mapInverseSlots();

        // Step 2. Reify slots with maximum cardinality
        reifySlotsWithMaxCardinality();

        // Step 3. Map cls to relations and views
        mapClsesToRelations();
        mapClsesToViews();

        // Step 4a. Map slots to attributes
        mapSlotsToAttributes();
        // Step 4b. Map slots to relations
        mapSlotsToRelations();
        // Step 4c. Map slots to foreign keys
        mapSlotsToForeignKeys();
}
```

Figure 5.5. JAVA code sample from KB2DB transformation outlining the algorithm step by step.

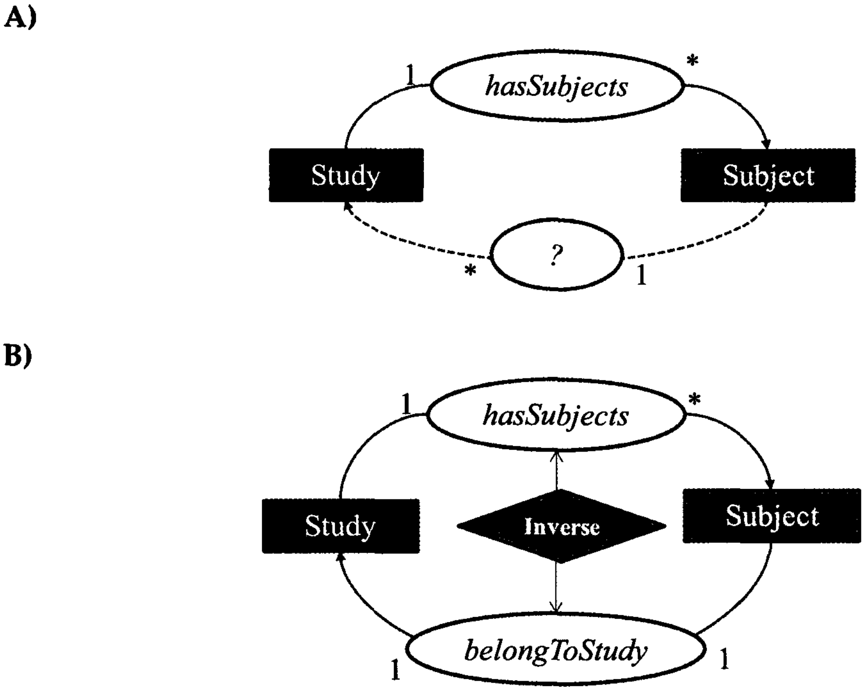


Figure 5.6. A) With only a single one-to-many directional relationship defined between *Study* and *Subject*, one can only safely assume that the inverse relationship from *Subject* to *Study* is also one-to-many. Hence, *Study* relates to *Subject* many-to-many. **B)** The existence of a one-to-one relationship from *Subject* to *Study* restricts that only one-to-many relationships exist between *Study* and *Subject*.

model (Stanford Center for Biomedical Informatics Research, 2010). Initially, if only *hasSubjects* is specified between classes *Study* and *Subject* in that direction as illustrated in Figure 5.6.A, each *Study* may have more than one *Subject*, however, we cannot infer if each *Subject* may only participate in a *Study.* In fact, the transformation program assumes a many-to-many relationship between the two classes. Defining an inverse slot adds specificity to the model as illustrated in Figure 5.6.B. The inverse relationship *belongToStudy* necessitates the constraint that a *Subject* may only participate in one *Study,* indicating a one-to-many relationship between *Study* and *Subject*.

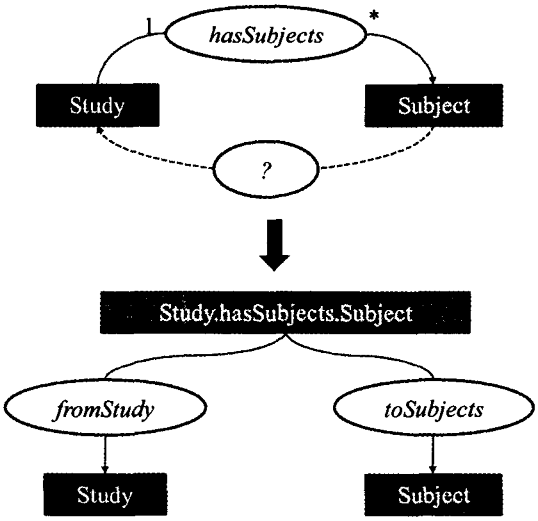


Figure 5.7. An example of reifying a one-to-many slot to an association entity. The lack of inverse relation can only allow us to safely conclude that Study relates to Subject in a many-to-many relationship. Therefore, an association is created in this normalization step.

This step of the transformation consolidates two slots defined by inverse slot relationships, keeping only the one with a stricter maximum cardinality. For the example above, slot *belongToStudy* is preserved for further transformation, while *hasSubjects* is hidden from the following steps. As a result of this step, reciprocal relationships between two objects are simplified to one.

Step 2. Map slots with maximum cardinality: This is a normalization step. In this step, slots with maximum cardinality greater than 1 are reified into actual entities for which each of the slot's maximum cardinality is no more than 1. The slot's value type makes no difference in this case, whether it is a primitive value type or an instance type. To illustrate this, I use the example from Figure 5.6.A where a directional one-to-many relationship slot exists between *Study* and *Subject*. The slot *hasSubjects* is reified to a class named *Study.hasSubjects.Subject* (as in Figure 5.7), following a convention of *[source class name].[relationship slot name].[target class name].* In the case of primitive target slot values, "target class name" is replaced with the name of the value type. This new association class then has two slots, one named *fromStudy* relating the new class back to the source class, e.g. *Study,* and the other named *toSubjects* relating the new class to the target class, e.g. *Subject.* After this step, the model has neither reciprocating relations nor one-to-many relations. Each slot in the model has a maximum cardinality of at most 1.

Step 3. Map Cls to Relations and Views: Here we implement rule ***T1.*** Each class is

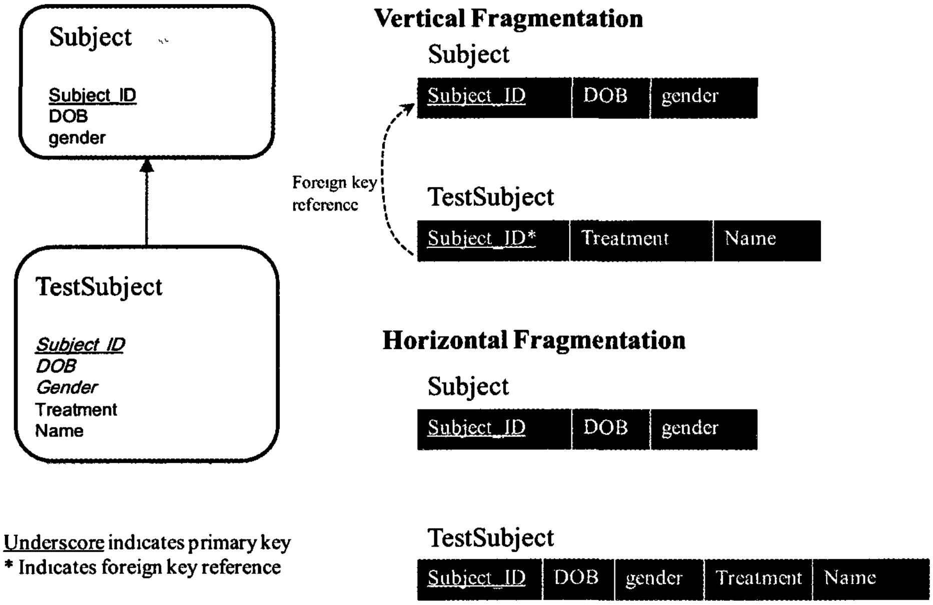


Figure 5.8. An example demonstrating the difference between vertical fragmentation and horizontal fragmentation. In vertical fragmentation, part of a child instance data would be inserted into both the parent table and part into the child table linked with a referencing key. In horizontal fragmentation, a child instance is inserted completely into the child table.

transformed to a table if it meets the following criteria: concrete, non-meta, non-system, and has at least one slot. Abstract classes do not have instances. Therefore, no table is created. Mappings between classes and tables are captured in *ModelMap.*

Class inheritance is transformed using horizontal fragmentation method. In an inheritance relationship, a child class inherits all its parent class's template slots. It can be more specialized than its parent by having additional template slots. Figure 5.8 shows an example of an inheritance relationship between classes *Subject* and *TestSubject.* The child class *TestSubject* inherits from *Subject* slots *Subject_ID, gender,* and *DOB.* Then *TestSubject* also has additional slots *Treatment* and N*ame.*

In vertical fragmentation, each class is transformed to a table. The child table contains only properties that were specific to the child and none of the inherited properties. It also has a foreign key reference to its parent table primary key. To store a tuple of data for the child table, inherited property data is inserted in the parent table. Then the parent tuple primary key is inserted into the child table along with any child-table specific data. As a result, the parent table is a superset of the child table data. *Subject_ID* is maintained to be unique in the parent table.

The vertical fragmentation transformation of inheritance is the most normalized form. However, the update operation of each child instance would require joins between two or more tables depending on the number of parents in the inheritance hierarchy. Horizontal fragmentation, although not perfectly normalized, optimizes access of data one object at a time. Getting access or update to a child object such as *TestSubject* requires no join. The unique primary ID for the object can be solely managed by the table itself.

```
      CREATE VIEW “v.TestSubject” AS
           SELECT “Subject_ID, “DOB”, “gender”, “Treatment”, “name”
           FROM “TestSubject”;

        CREATE VIEW “v.Subject” AS
           SELECT “Subject_ID”, “DOB”, “gender” FROM “Subject”
                UNION
           SELECT “Subject_ID”, “DOB”, “gender” FROM “v.TestSubject”;
```

Figure 5.9. View definition using examples from Figure 5.8. A view is created for each class. The view definition is a select union statement of the class itself and all its children classes’ view definitions.

Taking the horizontal fragmentation approach to implementing inheritance makes querying all instances of parent classes more difficult. For example, the *Subject* table in Figure 5.8 does not contain all members of its children’s tables. Therefore, a view is created for each class, regardless of whether the class is abstract or concrete. The view of a class is a select union statement of the class’s corresponding table (if concrete) and all of the class’s children’s views. Figure 5.9 shows the view definition for *Subject* and *TestSubject*. One can query all instances of a single type with a simple select statement from these pre-defined views. In Seedpod, this is a non-materialized view based on the view implementation of PostgreSQL.

Step 4. Map Slot to attribute, or association tables. Transformation of slots, rule **T2a,** is implemented in this step. At this point, as a result of step 2 above, all slots in the model (in memory) have maximum cardinalities no greater than 1. It is fairly straightforward that primitive type slots are transformed directly to RDB attributes. The *Attribute* is added to the corresponding table of the *Slot* container *Cls*. Slot descriptions are transformed into comments for attributes. Slot to attribute mappings are added to the *ModelMap* object. **T2c** is implemented in *Step 2* above. Method *mapSlotToRelations()* merely wraps up the step by creating a mapping in *ModelMap* from the slot to the new association table. Finally, slots of instance type and singular cardinality are transformed to foreign keys in the relational model (**T2b**). Appropriate mappings are created in the *ModelMap* between these slots and foreign keys.

5.3.3. Seedpod specific implementation

The previous section describes the general algorithm implemented for the transformation. In this section, a few Seedpod specific implementations in the transformation are described. The first one is transforming specific Protégé value types to PostgresSQL data value types. This is specific to the frame-based model and database that one chooses for the system. The second one is transforming in-line complex value types. In-line complex value types are used to accommodate complex compound data without creating new object classes.

Figure 5.10 lists Protégé types and their mappings to PostgresSQL data types. On the Protégé side, types other than *Any, Class,* and *Instance* are primitive types. There is no perfect one-to-one correlation between all of the types. The logic of the conversion is hard coded in the transformation program. Some of the mappings are strict such as Integer to Integer. The transformation employs user-specified RDB types as long as they do not violate allowed mapping rules. For example, user-specified RDB type *Integer* would be ignored if the Protégé type was *Boolean.* On the other hand, for RDB types with no corresponding Protégé types, such as *Date, Time, Timestamp,* and *Auto_increment,* user specification is honored over Protégé types. Protégé *Symbol* is transformed to *Varchar(n)*

KB_Type	DB_Type
Any	
Class	
Instance	Relation, Foreign Key
Float	Numberic
Integer	Integer
Boolean	Boolean
String	Varchar(n), text, character
Symbol	Varchar(n)
	Date
	Time
	Timestamp
	Auto_increment (serial4)

Figure 5.10. This table shows the value types conversion between a Protégé knowledge base and a relational database. Not all relational database types, which may vary depending on the database used, are listed here. Options listed are what are provided to the users as options in the Protégé user interface.

and *n* is determined based on the longest allowable symbol the user has entered. A default mapping exists for every Protégé value type if a user specification does not exist or make sense. For example, if the modeling user does not specify the specific DB type for a Protégé *String* type, then it is automatically transformed to *Varchar(255).*

Not all complex data types i.e. data types with multiple attributes, become first order classes in the Protégé model. A user may reuse these types by defining them only once. For example, *geolocations* involving *latitude* and *longitude* is represented as a complex data type class with two slots. This class is flagged in the model as an in-line data type. For example, a *hospital* has a *location* slot of instance type *geolocation.* Upon transformation, instead of creating a table for *geolocation* with a foreign key in *hospital* to *geolocation*, each of the slots in *geolocation* is inserted into the *hospital* table. The new attributes are renamed to *location.latitude* and *location.longitude.*

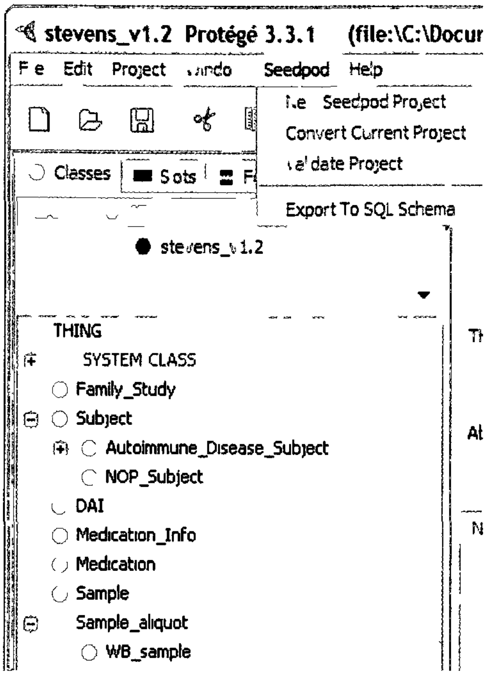


Figure 5.11. Protégé screenshot of Steven's lab model with an expanded Seedpod menu plug-in.

5.3.4. Executing Protégé2RDB

There are two ways to execute the transformation program. One is through a command prompt, running *Protege2RDB.Application* given three arguments: *[Path to Protégé Project] [RDB name] [Output file directory]*. The problem with this approach is that it assumes the input Protégé model is valid, modeled using Seedpod's meta-classes *:RDB_CLS* and *:RDB_SLOT*.

Users can also access the transformation capabilities through Protégé's graphical user interface (GUI). A Protégé project plug-in (Protege Developer Documentation) was developed to wrap the transformation functionalities to work in the Protégé GUI as a new Seedpod menu as shown in Figure 5.11. The menu contains the following functions:

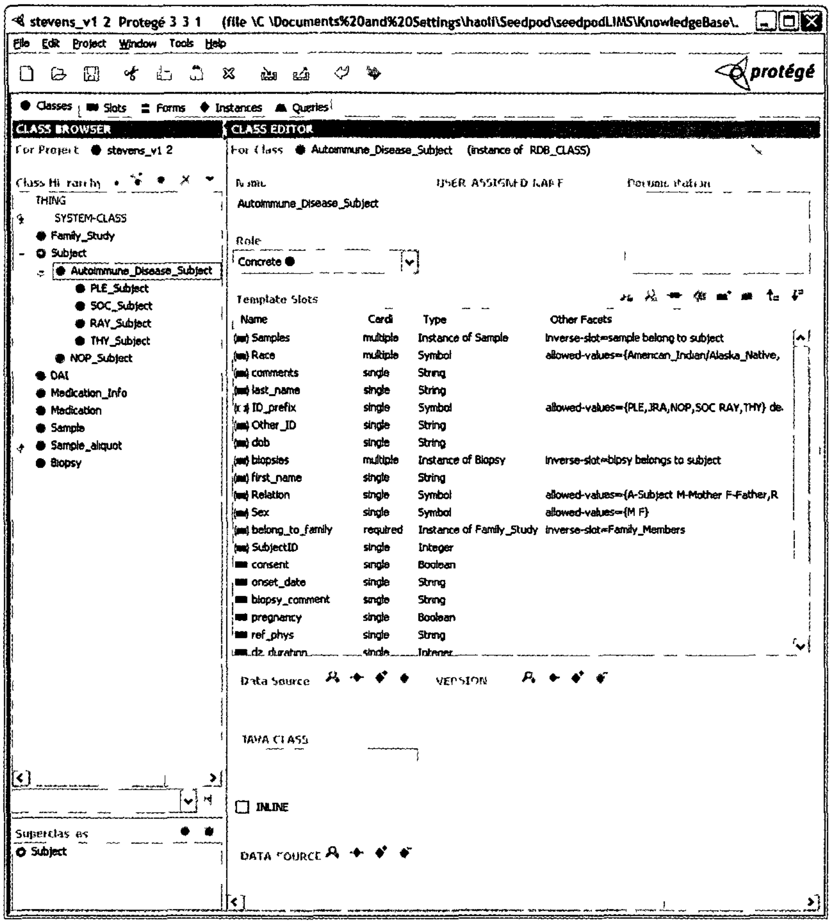
- Create new Seedpod projects with pre-built-in Seedpod meta-classes *:RDB_CLS* and *:RDB_SLOT.* The users will build models using these classes.
- Convert existing non-Seedpod projects into Seedpod projects. This converts all of the classes of type *:STANDARD_CLS* and *:STANDARD_SLOT* to *:RDB_CLS* and *:RDB_SLOT,* respectively.
- Validate existing project to ensure the classes use appropriate meta-classes.
- Transform and export current project. Transformation error and warnings messages are displayed at the end of the run.

5.4. Results

The transformation program *Protege2RDB* results in two SQL files: database schema definition and metadata table definition. Examples of these two files are shown in this section using examples from the experiment model developed for the Stevens Lab's lupus study. Figure 5.12 shows the model class hierarchy in the left panel with *Autoimmune_Disease_Subject* highlighted. The right panel center lists this class's template slots.

5.4.1. Output part 1: database definition

An *Rdb* class object is created as part 1 of the transformation. This object is serialized by a *RdbSchemaWriter* for a SQL output. However, if one wishes to write the schema to an UML format or generic XML format, one would just need to extend the generic class *RdbWriter* and implement the constructor and a *serialize(PrintStream)* function.



Name	Cardi	Type	Other Facets
(▬) Samples	multiple	Instance of Sample	inverse-slot=sample belong to subject
(▬) Race	multiple	Symbol	allowed-values={American_Indian/Alaska_Native,
(▬) comments	single	String	
(▬) last_name	single	String	
(r ‡) ID_prefix	single	Symbol	allowed-values={PLE,JRA,NOP,SOC RAY,THY} de.
(▬) Other_ID	single	String	
(▬) dob	single	String	
(▬) biopsies	multiple	Instance of Biopsy	inverse-slot=bipsy belongs to subject
(▬) first_name	single	String	
(▬) Relation	single	Symbol	allowed-values={A-Subject M-Mother F-Father,R
(▬) Sex	single	Symbol	allowed-values={M F}
(▬) belong_to_family	required	Instance of Family_Study	inverse-slot=Family_Members
(▬) SubjectID	single	Integer	
▬ consent	single	Boolean	
▬ onset_date	single	String	
▬ biopsy_comment	single	String	
▬ pregnancy	single	Boolean	
▬ ref_phys	single	String	
▬ dz_duration	single	Integer	

Figure 5.12. A screenshot of Steven's Lab Protégé model is shown here. This figure shows the template slot of the highlighted class *Autoimmune_Disease_Subject.* Template slots labeled with bracketed blue rectangular bricks are indicated as inherited slots from parent class, *Subject.* Regular blue rectangular bricks indicate this class's custom slots.

The class *Autoimmune_Disease_Subject* in Figure 5.12 is a concrete and non-leaf child class of *Subject.* According to transformation rule **T1a,** it is transformed to a table with the same name defined with the SQL statement shown in Figure 5.13. The table has a default primary ID. In the Seedpod system, this primary ID is a universal unique ID maintained by the *:Thing* table (more about the *:Thing* table in the next section). The table contains attributes that correspond to slots with simple types defined in the Protégé class, which includes inherited slots from the parent class, e.g. *first_name*. Each attribute has the same name as its slot.

```
CREATE TABLE "Autoimmune_Disease_Subject"
        ("ID" INTEGER,
        "comments" VARCHAR(50)DEFAULT NULL,--AUTO generated default value
        "Other_ID" VARCHAR(50)DEFAULT NULL,
        "SubjectID" INTEGER,
        "belong_to_family" INTEGER NOT NULL,
        "Relation" VARCHAR(12) DEFAULT NULL CHECK ("Relation" IN ('A-Subject', 'M-Mother', 'F-
Father', 'R1-Sibling_1', 'R2-Sibling_2', 'R3-Sibling_3' )),
        "first_name" VARCHAR(50) DEFAULT NULL,
        "ID_prefix" VARCHAR(3) DEFAULT NULL CHECK ("ID_prefix" IN ('PLE', 'JRA', 'NOP', 'SOC',
'RAY', 'THY' )),
        "Sex" VARCHAR(1) DEFAULT NULL CHECK ("Sex" IN ('M', 'F' )),
        "dob" DATE ,
        "last_name" VARCHAR(50) DEFAULT NULL,
        "pregnancy" BOOLEAN DEFAULT FALSE,
        "biopsy" BOOLEAN DEFAULT FALSE,
        "biopsy_comment" VARCHAR(50) DEFAULT NULL,
        "Disease" VARCHAR(4)DEFAULT NULL CHECK ("Disease" IN ('SLE', 'MCTD' )),
        "transfusion" BOOLEAN DEFAULT FALSE,
        "consent" BOOLEAN DEFAULT FALSE,
        "dz_duration" INTEGER ,
        "age_at_onset" VARCHAR(50)DEFAULT NULL,
        "ref_phys" VARCHAR(50) DEFAULT NULL,
        "onset_date" DATE,
        "birth_order" INTEGER DEFAULT 1,
        "dx_date" DATE,
         PRIMARY KEY ("ID"));
```

Figure 5.13. Sample SQL table definition of the transformation for class *Autoimmune_Disease*_Subject shown in Figure 6.13. Comments auto-generated at the end of each line, such as "-- AUTO generated default value", are deleted for visual clarity. Names of tables and attributes are in quotes to preserve the original Protégé model's capitalization and space. This implementation is specific for PostgreSQL. Some other database may use other characters for the same purpose.

```
CREATE VIEW "v.Autoimmune_Disease_Subject" AS
        SELECT "ID", "first_name", "last_name", "dob", "SubjectID", "belong_to_family", "comments",
        "Sex", "Relation", "ID_prefix", "Other_ID", "biopsy_comment", "transfusion", "pregnancy",
        "age_at_onset", "dx_date", "ref_phys", "dz_duration", "Disease", "birth_order", "biopsy",
        "consent", "onset_date"
        FROM "Autoimmune_Disease_Subject"
UNION
        SELECT "ID", "first_name", "last_name", "dob", "SubjectID", "belong_to_family", "comments",
        "Sex", "Relation", "ID_prefix", "Other_ID", "biopsy_comment", "transfusion", "pregnancy",
        "age_at_onset", "dx_date", "ref_phys", "dz_duration", "Disease", "birth_order", "biopsy",
        "consent", "onset_date"
        FROM "v.PLE_Subject"
UNION
        SELECT "ID", "first_name", "last_name", "dob", "SubjectID", "belong_to_family", "comments",
        "Sex", "Relation", "ID_prefix", "Other_ID", "biopsy_comment", "transfusion", "pregnancy",
        "age_at_onset", "dx_date", "ref_phys", "dz_duration", "Disease", "birth_order", "biopsy",
        "consent", "onset_date"
        FROM "v.SOC_Subject"
UNION
        SELECT "ID", "first_name", "last_name", "dob", "SubjectID", "belong_to_family", "comments",
        "Sex", "Relation", "ID_prefix", "Other_ID", "biopsy_comment", "transfusion", "pregnancy",
        "age_at_onset", "dx_date", "ref_phys", "dz_duration", "Disease", "birth_order", "biopsy",
        "consent", "onset_date"
        FROM "v.RAY_Subject"
UNION
        SELECT "ID", "first_name", "last_name", "dob", "SubjectID", "belong_to_family", "comments",
        "Sex", "Relation", "ID_prefix", "Other_ID", "biopsy_comment", "transfusion", "pregnancy",
        "age_at_onset", "dx_date", "ref_phys", "dz_duration", "Disease", "birth_order", "biopsy",
        "consent", "onset_date"
        FROM "v.THY_Subject";
```

Figure 5.14. This is an example of a view definition for a non-leaf concrete class in SQL for *Autoimmune_Disease_Subject.*

An attribute type is determined according to mappings listed in Figure 5.10. For example, *SubjectID* has type Integer. Type Varchar(n) has default length, or n, of 50. The transformation adds a comment when this default is used. Attribute *Sex* is an example of transformation of Protégé type Symbol to SQL type Varchar(n). Symbols in a Protégé model can have predefined values such as *"M"* and *"F"* for male and female in this example. Therefore, type Symbol becomes type Varchar(1) since 1 is the longest symbol option. A

```
ALTER TABLE "Autoimmune_Disease_Subject"
        ADD CONSTRAINT "fk_belong_to_family" FOREIGN KEY
        ("belong_to_family")
REFERENCES "Family_Study" ON DELETE CASCADE ;
```

Figure 5.15. Here is an example of a foreign key referential integrity constraint definition. Foreign key *belong_to_family* represents a relationship from *Autoimmune_Disease_Subject* to *Family_Study.*

check constraint follows that limits the value input such as `CHECK ("Sex" IN ('M', 'F'))`. *Relation* is another example of Symbol type.

According to rule **T1b**, a view is created. An example of the view definition for *Autoimmune_Disease_Subject* can be found in Figure 5.14. Because the transformation chooses to use horizontal fragmentation to represent inheritance, the user cannot query from all instances of autoimmune disease subjects if this view is not defined. Finally, this view is stored as a select statement and never materialized in PostgreSQL. Depending on the specific database engine used, this view may or may not be materialized.

Slots of instance type define relationships between the container class and the destination class. For example, *belong_to_family* is a relationship slot in *Autoimmune_Disesase_Subject.* It is an inverse relationship of *Family_Members* in class *Family.* Following implementation step 1, *Family_Members* is hidden since *belong_to_family* is sufficient to describe this relationship. A foreign key with the same name of Integer type is defined in the *Autoimmune_Disease_Subject* table. Its referential integrity is established after all the tables are defined with an ALTER TABLE statement (Figure 5.15).

A slot with multiple value cardinality represents a one-to-many relationship regardless of whether the value type is an instance or a simple type. Slot *Race* is an example of that. Each subject can have one or more race descriptors. The value type for *Race* is symbol, which is transformed to type varchar(n). Figure 5.16 shows the SQL statement that creates a table for the relationship between *Autoimmune_Disease_Subject* and *race*. This table contains a foreign key to the subject table. Each subject ID may be associated with one or more race symbol constrained to a list of options. Each tuple in this table contains a unique pair of subject ID and race symbol, which is defined as a combination primary key.

```
CREATE TABLE "Autoimmune_Disease_Subject.Race.Symbol"(
    "TO.Symbol.403" VARCHAR(32) NOT NULL
        CHECK ("TO.Symbol.403" IN ('American_Indian/Alaska_Native',
            'African_American/Black','Native_Hawaiian/Pacific_Islander, 'Hispanic',
            'Asian', 'Caucasian', 'Other')),
    "FROM.Autoimmune_Disease_Subject.404" INTEGER NOT NULL,
    PRIMARY KEY (
     "TO.Symbol.403",
        "FROM.Autoimmune_Disease_Subject.404")
);

ALTER TABLE "Autoimmune_Disease_Subject.Race.Symbol"
    ADD CONSTRAINT "fk_FROM.Autoimmune_Disease_Subject.404"
    FOREIGN KEY ("FROM.Autoimmune_Disease_Subject.404")
    REFERENCES "Autoimmune_Disease_Subject" ON DELETE CASCADE;
```

Figure 5.16. An example SQL statement showing creation of an association table for a one-to-many relationship called *race* between *Autoimmune_Disease_Subject* and *Symbol.*

A more complex many-to-many relationship transformation is illustrated by slot *Samples* which has Instance type of class *Sample* in Figure 5.13. Each subject can have multiple samples collected for him. After the source and target tables are defined,

```
CREATE TABLE "Autoimmune_Disease_Subject.Samples.Sample"(
    "TO.Sample.406" INTEGER  NOT NULL,
    "FROM.Autoimmune_Disease_Subject.407" INTEGER NOT NULL,
    PRIMARY KEY(
        "TO.Sample.406",
        "FROM.Autoimmune_Disease_Subject.407"
    )
);

ALTER TABLE "Autoimmune_Disease_Subject.Samples.Sample"
    ADD CONSTRAINT "fk_TO.Sample.406"
    FOREIGN KEY ("TO.Sample.406")
    REFERENCES "Sample" ON DELETE CASCADE;

ALTER TABLE "Autoimmune_Disease_Subject.Samples.Sample"
    ADD CONSTRAINT "fk_FROM.Autoimmune_Disease_Subject.407"
    FOREIGN KEY ("FROM.Autoimmune_Disease_Subject.407")
    REFERENCES "Autoimmune_Disease_Subject" ON DELETE CASCADE;
```

Figure 5.17. A many-to-many relationship is transformed to an association table with foreign keys that reference back to source and target tables.

Autoimmune_Disease_Subject and *Sample* respectively, an association table called *Autoimmune_Disease_Subject.Samples.Sample* is defined consisting of two foreign keys referencing the source and target table primary keys. Unique pairs of the two foreign keys make up the primary key for this association table. Figure 5.17 shows the SQL definition statements.

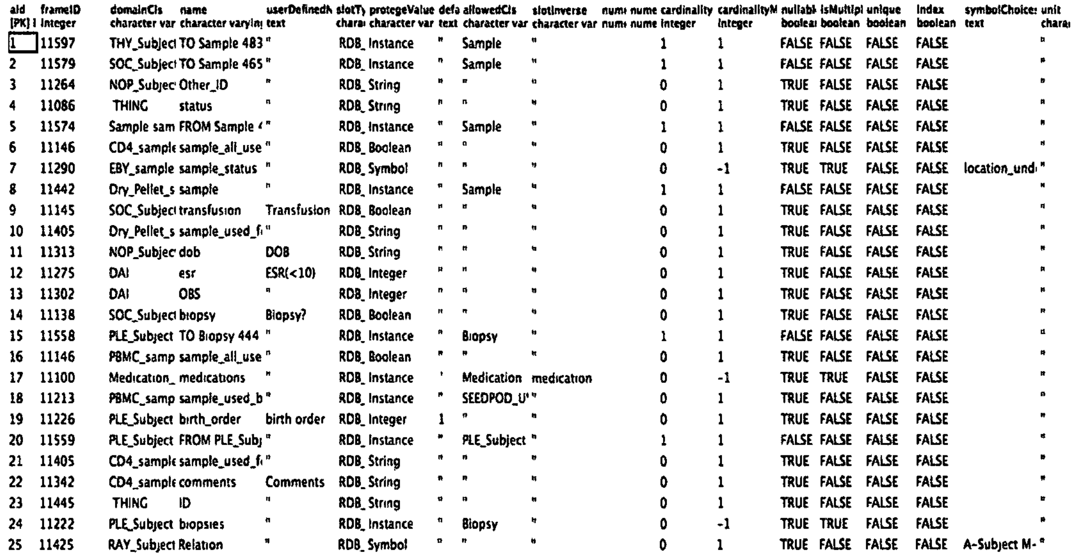
5.4.2. Output part 2: Mapping meta-data

The second output of the transformation is mappings between the frame-based model and the relational model. It is called *ModelMap* in the JAVA transformation program. This mapping allows programs that use this mapping information to reconstruct the source and target models and mappings between them. Detailed demonstration of the usage is described in Chapter 4. This *ModelMap* object is serialized into SQL statements with examples shown in Figure 5.18.

Class mappings metadata is stored in the *:RDB_CLASS* table while slot to attribute mapping data is stored in the *:RDB_ATTRIBUTE* table. Figure 5.19 and Figure 5.20 illustrate

```
INSERT INTO ":RDB_CLASS" (
    "cid", "frameID", "name", "userDefinedName", "clsType", "parent", "primaryKey",
    "inline", "isConcrete", "documentation", "browserPattern", "tableName",
    "viewName", "javaClass")
VALUES (
    DEFAULT , 11515, $sp$Autoimmune_Disease_Subject.Samples.Sample$sp$,
    DEFAULT , $sp$:RDB_CLASS$sp$, $sp$:REIFIED_SLOT_CLS$sp$, $sp$$sp$, false,
    true, $sp$$sp$, $sp$Autoimmune_Disease_Subject.Samples.Sample VAL(id)$sp$,
    $sp$Autoimmune_Disease_Subject.Samples.Sample$sp$, DEFAULT , DEFAULT );

      ...
INSERT INTO ":RDB_ATTRIBUTE" (
    "aid", "frameID", "domainCls", "name", "userDefinedName", "slotType",
    "protegeValueType", "defaultValues", "allowedCls", "slotInverse",
    "numericMin", "numericMax", "cardinalityMin", "cardinalityMax", "nullable",
    "isMultiple", "unique", "index", "symbolChoices", "unit", "documentation",
    "rdbAttributeName", "rdbTarget", "dbValueType", "dbValueLength",
    "isAssociated", "expression", "viewSequence", "formWidget",
    "formWidgetParam", "viewWidget", "viewWidgetParam")
VALUES (
    DEFAULT , 11227, $sp$Subject$sp$, $sp$ID_prefix$sp$, $sp$id_prefix$sp$,
    $sp$:RDB_ATTRIBUTE$sp$, $sp$Symbol$sp$, $sp$$sp$, $sp$$sp$, $sp$$sp$, NULL
    , NULL , 0, 1, true, false, false, false, $sp$PLE    JRA    NOP    SOC    RAY
       THY$sp$, $sp$$sp$, $sp$$sp$, $sp$ID_prefix$sp$,
    $sp$:RDB_ATTRIBUTE(ID_prefix)$sp$, $sp$VARCHAR$sp$, 3, true, $sp$$sp$, 0.0,
    $sp$SELECT$sp$, $sp$3$sp$, $sp$STRING$sp$, $sp$$sp$);
```

Figure 5.18. Examples of *ModelMap* serialization in SQL statements. These statements insert data tuples into the *:RDB_CLASS* and *:RDB_ATTRIBUTE* classes respectively. ("$sp$" is used to mark the beginning and end of a string instead of double or single quotes.)



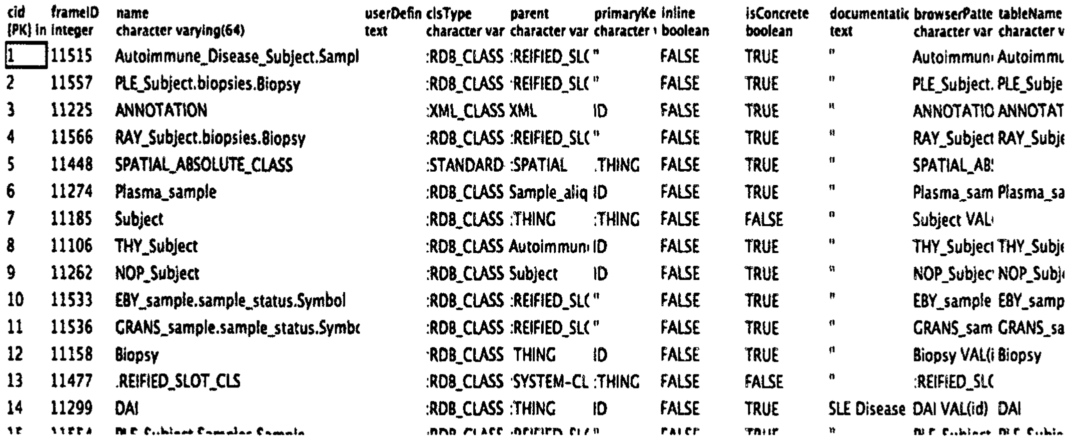
aId	frameID	domainCls	name	userDefinedN	slotTy	protegeValue	defa	allowedCls	slotInverse	num	nume	cardinality	cardinalityM	nullabl	isMultipl	unique	index	symbolChoice	unit
[PK] I	Integer	character var	character varying	text	chara	character var	text	character var	character var	num	nume	integer	integer	boolea	boolean	boolean	boolean	text	chara
1	11597	THY_Subject	TO Sample 483	"	RDB_	Instance	"	Sample	"			1	1	FALSE	FALSE	FALSE	FALSE		"
2	11579	SOC_Subject	TO Sample 465	"	RDB_	Instance	"	Sample	"			1	1	FALSE	FALSE	FALSE	FALSE		"
3	11264	NOP_Subjec	Other_ID	"	RDB_	String	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
4	11086	THING	status	"	RDB_	String	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
5	11574	Sample sam	FROM Sample 4	"	RDB_	Instance	"	Sample	"			1	1	FALSE	FALSE	FALSE	FALSE		"
6	11146	CD4_sample	sample_all_use	"	RDB_	Boolean	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
7	11290	EBY_sample	sample_status	"	RDB_	Symbol	"	"	"			0	-1	TRUE	TRUE	FALSE	FALSE	location_und	"
8	11442	Dry_Pellet_s	sample	"	RDB_	Instance	"	Sample	"			1	1	FALSE	FALSE	FALSE	FALSE		"
9	11145	SOC_Subject	transfusion	Transfusion	RDB_	Boolean	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
10	11405	Dry_Pellet_s	sample_used_f	"	RDB_	String	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
11	11313	NOP_Subjec	dob	DOB	RDB_	String	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
12	11275	DAI	esr	ESR(<10)	RDB_	Integer	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
13	11302	DAI	OBS	"	RDB_	Integer	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
14	11138	SOC_Subject	biopsy	Biopsy?	RDB_	Boolean	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
15	11558	PLE_Subject	TO Biopsy 444	"	RDB_	Instance	"	Biopsy	"			1	1	FALSE	FALSE	FALSE	FALSE		"
16	11146	PBMC_samp	sample_all_use	"	RDB_	Boolean	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
17	11100	Medication_	medications	"	RDB_	Instance	'	Medication	medication			0	-1	TRUE	TRUE	FALSE	FALSE		"
18	11213	PBMC_samp	sample_used_b	"	RDB_	Instance	"	SEEDPOD_U	"			0	1	TRUE	FALSE	FALSE	FALSE		"
19	11226	PLE_Subject	birth_order	birth order	RDB_	Integer	1	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
20	11559	PLE_Subject	FROM PLE_Subj	"	RDB_	Instance	"	PLE_Subject	"			1	1	FALSE	FALSE	FALSE	FALSE		"
21	11405	CD4_sample	sample_used_f	"	RDB_	String	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
22	11342	CD4_sample	comments	Comments	RDB_	String	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
23	11445	THING	ID	"	RDB_	String	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE		"
24	11222	PLE_Subject	biopsies	"	RDB_	Instance	"	Biopsy	"			0	-1	TRUE	TRUE	FALSE	FALSE		"
25	11425	RAY_Subject	Relation	"	RDB_	Symbol	"	"	"			0	1	TRUE	FALSE	FALSE	FALSE	A-Subject M-	"

Figure 5.19. Screenshot of the *:RDB_ATTRIBUTE* table from PostgresAdmin.

these metadata tables with screenshots of the tables filled with data from PostgresAdmin. The table :*RDB_CLASS* shows information of classes from the Protégé model and what they are mapped to after transformation in the relational model. For example, a class name is mapped to table name and a view name in the same tuple. Similarly, the table *:RDB_SLOT* contains metadata about each slot mapping to attribute, such as cardinality, allowed values, user defined names, attribute names, database types, etc. This metadata information is necessary for constructing queries to the data object tables. Refer to Chapter 4 for detailed usage discussion.

5.5. Conclusion

This chapter covered the theoretical and practical aspects of transforming a frame-based model to a relational model. The definition of the two models and transformation



cid {PK} in	frameID integer	name character varying(64)	userDefin text	clsType character var	parent character var	primaryKe character	inline boolean	isConcrete boolean	documentatic text	browserPatte character var	tableName character v
1	11515	Autoimmune_Disease_Subject.Sampl		:RDB_CLASS	:REIFIED_SL(	"	FALSE	TRUE	"	Autoimmun	Autoimmu
2	11557	PLE_Subject.biopsies.Biopsy		:RDB_CLASS	·REIFIED_SL(	"	FALSE	TRUE	"	PLE_Subject.	PLE_Subje
3	11225	ANNOTATION		:XML_CLASS	XML	ID	FALSE	TRUE	"	ANNOTATIO	ANNOTAT
4	11566	RAY_Subject.biopsies.Biopsy		:RDB_CLASS	:REIFIED_SL(	"	FALSE	TRUE	"	RAY_Subject	RAY_Subj
5	11448	SPATIAL_ABSOLUTE_CLASS		:STANDARD	:SPATIAL	.THING	FALSE	TRUE	"	SPATIAL_AB!	
6	11274	Plasma_sample		:RDB_CLASS	Sample_aliq	ID	FALSE	TRUE	"	Plasma_sam	Plasma_sa
7	11185	Subject		:RDB_CLASS	:THING	:THING	FALSE	FALSE	"	Subject VAL	
8	11106	THY_Subject		:RDB_CLASS	Autoimmun	ID	FALSE	TRUE	"	THY_Subject	THY_Subj
9	11262	NOP_Subject		:RDB_CLASS	Subject	ID	FALSE	TRUE	"	NOP_Subjec	NOP_Subj
10	11533	EBY_sample.sample_status.Symbol		:RDB_CLASS	:REIFIED_SL(	"	FALSE	TRUE	"	EBY_sample	EBY_samp
11	11536	GRANS_sample.sample_status.Symb(		:RDB_CLASS	:REIFIED_SL(	"	FALSE	TRUE	"	GRANS_sam	GRANS_sa
12	11158	Biopsy		·RDB_CLASS	THING	ID	FALSE	TRUE	"	Biopsy VAL(i	Biopsy
13	11477	.REIFIED_SLOT_CLS		:RDB_CLASS	·SYSTEM-CL	:THING	FALSE	FALSE	"	:REIFIED_SL(	
14	11299	DAI		:RDB_CLASS	:THING	ID	FALSE	TRUE	SLE Disease	DAI VAL(id)	DAI
15	11554	PLE_Subject.Samples.Sample		:RDB_CLASS	:REIFIED_SL(	"	FALSE	TRUE	"	PLE_Subject	PLE_Subje

Figure 5.20. Screenshot of the *:RDB_CLASS* table from PostgresAdmin

rules in the first two sections provide a generalized methodology. The algorithm is then tested in Seedpod with a JAVA program that transforms a Protégé frame-based model to relational database definition that could be executed successfully in a PostgreSQL database. Detailed implementation steps are described, including additional Seedpod system specific implementation to bridge the difference between the two models. The transformation results in a database definition and metadata written in SQL. The resulting SQL statements can be executed in relational database management systems such as PostgreSQL.

The transformation algorithm is generic. The transformation program for a specific frame-based model interface, such as Protégé, only needs to be built once. All future transformations can be done fully automatically. Within the scope of Seedpod, this allows the system to take advantage of Protégé's modeling GUI in addition to the data storage power of a relational database. The actual usability of the resulting relational database and completeness of the model translation are evaluated critically in Chapter 6.

6. CRITICAL ANALYSIS

LIMS requirements that are pertinent to this thesis project are discussed in Chapter 2. The five requirements are as follows:

R1. The system must allow scientific users to manage large and complex datasets for ease of retrieval and organization. Data may be multimedia with metadata. Data may also have complex relationships.

R2. The system must support remote data management, allowing multiple users and multiple disciplines to work together.

R3. The system must support scientists to get involved and contribute in the process of the system design, development and testing process.

R4. The system must keep development time, effort, and cost low.

R5. The system should lower the complexity to deal with system evolution.

Various existing solutions are evaluated against this set of requirements in Chapter 3. In this chapter, Seedpod is evaluated against the same set of requirements. Seedpod implementations for the Stevens Lab's Lupus Research Lab (LRL) and Ojemann's Single Unit Recording Lab (SUR) are used as examples throughout the chapter. Additional evaluation notes are made about the system which point to directions of future work on Seedpod.

6.1. Two Seedpod LIMS examples

Both Ojemann's SUR and Steven's LRL are described in Section 2.1. Their data management needs, both technical and social, were distilled to the requirements above. Seedpod is not designed to be the best all around LIMS but to meet these requirements using a model-driven approach. Seedpod used SUR as a motivating problem throughout its version one development and testing. When Seedpod was more mature in its second iteration development, it was applied to LRL for testing.

6.2. Evaluation against the requirements

The author developed both implementations of Seedpod to LRL and SUR with no real-world users. The following evaluation is therefore based on personal critical opinion of the design, usage, and performance of Seedpod.

6.2.1. R1

> *The system must allow scientific users to manage large and complex datasets for ease of retrieval and organization. Data may be multimedia with metadata. Data may also have complex relationships.*

In lieu of a flexible XML data store such as in Teranode, Seedpod opted for using a relational SQL database. Both storing and retrieval of large datasets are robust, efficient, and fast. The technology has been well tested in the past two decades in commercial products and scientific products.

SUR collects patient clinical data, time series data from multiple electrodes, and surgical photos. To accommodate multimedia data, Seedpod's database manages metadata for multimedia data, such as file name, author, and a pointer to the actual data file. The actual data is stored as files and managed by a file system. A user does not have to manually manage the physical file structure, but instead gains the ease of accessing the file via metadata stored in the database. Multimedia data is integrated with other numerical or textual data without breaking a workflow. This technique has been used by many information management systems.

The use of Protégé for modeling LIMS is to provide ease in modeling complex relationships between data objects. Users do not need to be concerned with the actual implementation of the relationship. These relationships may be hierarchical parent-child relationships, containment relationships, or complex relationships with attributes. For example, LRL has several different patient subjects. The class definition is easily re-used by using a hierarchical structure to organize its control subject and various experiment protocol subjects.

For the most part, Seedpod satisfies this requirement for the purpose of data entry and some data retrieval. Its web-based GUI allows the user to manage multimedia data along with tabular data in an object-oriented fashion. Relationships between objects provide navigational workflow between the pages in the web-based GUI. However, since Seedpod cannot anticipate how users would need to retrieve data for analysis or complex visualization, it does not come with pre-packaged SQL join queries. These join queries are highly custom for each application. They also require someone that is well versed in writing relational queries. Then a custom web page for the visualization would need to be implemented. Seedpod is mostly concerned with getting data in and not data analysis and complex visualization.

6.2.2. R2

> *The system must support remote data management, allowing multiple users and multiple disciplines to work together.*

Seedpod's server application and database should be installed on a secured server. Users can enter or access data from anywhere with an internet access. Multiple users can add or modify data at the same time without worrying about data files out of synch, because the content of the web pages is dynamically generated from the shared database. For example, LRL consists of scientists in two locations: Seattle Children's Hospital and UW South Lake Union Lupus Laboratory. The former collects clinical data and the latter provides wet lab data. They need to share patient information and ultimately combine the data for analysis. Remotely managing and accessing up-to-date data would reduce data error and the inconvenience of manually synching data.

A Seedpod system user belongs to one of the three user groups with certain privileges. For example, an administrator user can have all data access and the ability to add other users. A power user can edit all data. A collaborator can read data only. A collaborator may be someone from outside of the lab that would like to share and access the data.

While Seedpod satisfies this requirement, its implementation for different user access is only implemented for the purpose of demonstration. A working system may require the ability to allow an administrator to create new user groups and manage data access for the different groups. Additionally, it has become increasingly important in the scientific research community to track the provenance of data, which is meta-data about how each piece of data has evolved in the process from collection to analysis. The complex nature of the problem is beyond the scope of this project.

6.2.3. R3

The system must support scientists to get involved in and contribute to the process of the LIMS design, development and testing process.

Scientific users are more knowledgeable about the data they collect and the domain they study. Therefore, they may be more adequate in modeling the LIMS. As described in Seedpod's development workflow in 4.7, scientists participate in steps 1 and 3 of the development process for modeling and testing. The caveat is that the modeling environment may not be the most intuitive interface or best choice of expressive language. If they are not familiar with the modeling environment, they could learn to read the model for accuracy while working with informaticists to develop the model. While we were working with the graduate students in SUR, they were able to check the Protégé model for correctness. Communications about the data model between scientists also started to clear up when people could use the same vocabulary in the model. Instead of an illegible relational database DDL document, scientists may feel more in control of the development process by using the more intuitive graphical modeling environment. Scientists can feel more involved working with the informaticists.

Seedpod uses Protégé, which is a knowledge-base management tool, for modeling. The usability of the tool's GUI is debatable. However, observation and experience of working with several scientists point to the fact that scientists are very willing to learn to use Protégé and find the modeling concepts easily understandable.

The real hurdle of using Seedpod is that one may not be able to see how the system works or if the LIMS requires tweaking until one has gone through the three steps described in 4.7. What Seedpod needs is an interactive development environment (IDE) that provides previews and debugging tools to help the modeling user see what the resulting system GUI would look like while working on the model. This IDE would function as an emulator, allowing users to see affects of changes made to the LIMS model. For example, it would be helpful to the users to see the difference between the different GUI widgets. Development of an IDE can only be worthwhile as a next step research and development after the whole Seedpod system has been shown to provide value.

6.2.4. R4

The system must keep development time, effort, and cost low.

Seedpod is built using only open-source technology, which includes the modeling environment Protégé, the relational database PostgreSQL, and the web server Tomcat. In terms of software and hardware, a PI would only need to pay for the computing instrument that houses the web server and database server.

The development process is shorter using Seedpod from modeling to deployment. The time passed between scientists testing and informaticists debugging can be short for this quick iterative development cycle. It took the author 2-3 hours to interview the LRL scientists in 3 interview sessions, and then less than 2 hours to create the initial data model in Protégé. A majority of the time is spent on developing the model and getting it right. Setting up the system to auto-generate the relational database and then deploy the web service is simple and straight forward. Additional time may be needed to debug the model and customize GUI widgets.

From the perspective of an informatics team, Seedpod is a system that can be adapted for multiple laboratories' data management needs. Various laboratory implementations of Seedpod differ only in their models. More effort can be spent on customizing Seedpod for specific needs. The server application and model transformation pieces of Seedpod remain the same for both SUR and LRL Seedpod applications. The major difference is in the starting Protégé models.

A traditional web-based application development team such as SIG would consist of someone with domain knowledge, an expert in relational database, a system admin, and a web application software engineer. Seedpod requires far less expertise and knowledge for it to deploy, which means the system of complex computing tools behind it is made available to more naive users with little or no computing background. In essence, Seedpod drastically lowers the threshold to adopt and develop a new LIMS.

Seedpod satisfies this requirement for the most part. However, customization of Seedpod may or may not be an expensive operation. For example, customizing a visualization widget for SUR time series data is not a trivial task. Seedpod supports the widget development with a simple plug-in frame work. A software engineer would then need to write a piece of server-side code that implements the *Widget* programming interface. Then the widget would need to fetch the time series file and render an image in for the web. This process requires the engineer to be very familiar with Seedpod.

6.2.5. R5

The system should lower the complexity to deal with system evolution.

There are three approaches for evolving Seedpod. The first approach is to make changes directly on the model, then re-interpret the model changes into changes for the database schema, data in the database, and application. Seedpod's server application is completely model-independent, which means regardless of changes to the model the server application does not need to be changed. The database definition is auto-generated from the model. In order for Seedpod to evolve seamlessly, it needs the ability to translate Protégé model changes into relational database changes. Changes involving changing the webapp widgets are straightforward. Changes involving changing the data table structures, such as adding an attribute, are more involved. Techniques for evolving relational databases can be incorporated (Hick & Hainaut, 2003; Dominguez, Lloret, & Rubio, 2002).

Another approach to evolving MDA systems follows the principals that encourage an agile data warehouse to allow users "easily ingest, digest, produce and adapt data at a rapid pace (Cohen, Dolan, Dunlap, Hellerstein, & Welton, 2009)." A need for Seedpod to evolve comes up when a new experiment protocol is developed. It may make more sense to create a separate data management system instead of changing the existing one. Within the same laboratory, an existing model may be reused and modified to create a new model. For example, LRL model definition for *Subject* can be reused when a new protocol subject by creating a new child class of *Subject* as shown in Figure 6.1. The database server and web application server for Seedpod can both be reused with small modifications to connectivity configuration. Following this approach, the scientific user can get started with collecting new data quickly. When one needs to analyze the new database with the older database(s), integration techniques such as mediators or distributed database management systems can be used (Ludäscher, Gupta, & Martone, 2003; Tang, Kadiyska, Li, Suciu, & Brinkley, 2003; Hachem, Gennert, & Ward, 1993).

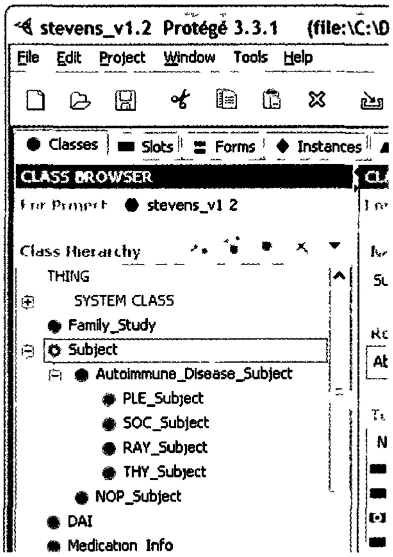

stevens_v1.2 Protégé 3.3.1 (file:\C:\D
File Edit Project Window Tools Help
● Classes ▬ Slots ☰ Forms ◆ Instances
CLASS BROWSER
For Project ● stevens_v1 2
Class Hierarchy
THING
SYSTEM CLASS
● Family_Study
○ Subject
● Autoimmune_Disease_Subject
● PLE_Subject
● SOC_Subject
● RAY_Subject
● THY_Subject
● NOP_Subject
● DAI
● Medication Info

Figure 6.1. LRL implements several types of subjects for various experiment protocols. When more subjects are needed for new experiments, a new class can be added as a child to *Autoimmune_Disease_Subject.*

	R1 Data management features	R2 Multi-user remote access	R3 Scientist user involvement	R4 Low development time and technical cost	R5 Ease of system evolution	Total
S1: Custom solutions	3	3	1	1	1	9
S2: COTS (Excel/Affymetrix)	2 (1/3)	2 (1/3)	2 (3/1)	1 (1/1)	2 (3/1)	9
S3: Tool kits	3	3	1	2	1	10
S4: Model-driven	2	3	2	3	3	13
Seedpod	1	3	2	3	3	12

Figure 6.2. Comparing Seedpod to existing solutions extending Figure 3.9 in Section 3.5.

Finally, the third approach takes a middle ground from the previous approaches. The user first creates a new model that would work for the new data. Seedpod can help to auto-generate a new database from the model. Then a data engineer would apply data integration techniques to export data from the old database and import into the new database. Again, nothing needs to be done to Seedpod's server application.

Seedpod does not solve the evolution problem but it has shrunken a big part of the problem with its model-independent server application. The above three approaches are worthy of investigating for future work.

6.3. Conclusion

Figure 6.2 extends Figure 3.9 to include Seedpod in comparison with existing solutions. Seedpod performs similarly to existing MDA solutions in meeting R2, R3, and R4, but falls short in its features (R1). Seedpod was developed by one graduate student compared to teams of experienced engineers. Both Teranode and Portofino support advanced workflow modeling and management which Seedpod does not. All three MDA solutions ease the complexity of system evolution but that is speculative.

Seedpod is a prototype that has been evaluated based on the author's critical analysis against requirements listed in Chapter 2. Seedpod meets R1-R4 for the most part, and lays down the foundation for R5. Seedpod needs to be evaluated with real world problems and users. It may then mature through more iterations of refinement.

7. CONCLUSION

This thesis has described a model-driven LIMS called Seedpod. It is an approach to building LIMS using a formal knowledge model. A methodology was developed to automatically transform a knowledge model in Protégé to a relational model. The resulting LIMS is a web application that dynamically generates the web-based GUI using the knowledge model and meta-data from the transformation. The web application allows users to manage and browse data that is stored in a database. A plug-in framework allows developers to extend and customize Seedpod.

Seedpod has the ability to manage large complex multimedia data sets. Users can access data anywhere with an internet access. The methodology encourages the users to work closely with the developers on modeling the LIMS. The resultant cost-saving LIMS can be quickly developed by simply creating a Protégé model and without writing any program code. In the future, Seedpod may lower the burden of system evolution by allowing the user to only make changes to the LIMS model. This chapter concludes the thesis with its contributions and future work.

7.1. Contributions

1) *Knowledge-model-driven approach to building LIMS*: Very few MDA LIMS exist. None of them uses a formal knowledge model to represent the LIMS. This project uses an open-source knowledge model developed using Protégé to

capture information about a LIMS. Naïve users without programming skills can create a LIMS with a relational database and a web application by simply creating a descriptive model. The knowledge model is machine-readable; the rest of the LIMS components are driven by the model through either automatic code translation or querying of the model. The use of a formal knowledge model opens the opportunity to sharing and ease of integration with other knowledge bases.

2) *Automatic transformation of Protégé model to relational model*: A methodology is developed to automatically transform the knowledge model in Protégé to a relational model written in standard SQL data definition language (DDL). The resulting DDL can be used directly to create a relational database. This automatic translation is fast and separates the user from the technical complexity of developing a relational data model from the database. Changing the model during development using the Protégé GUI is much easier than making changes to a relational database DDL.

3) *Domain independent LIMS:* The LIMS web application is domain independent. In other words, it can be deployed for various laboratories in different research studies. The content of the LIMS is provided and informed by the knowledge model and meta-data from the transformation. As demonstrated by Figure 7.1, SUR and LRL each have their own models. The transformation and application components merely query the model and the database without any laboratory specific code. The LIMS engine is built once but can be used many times.

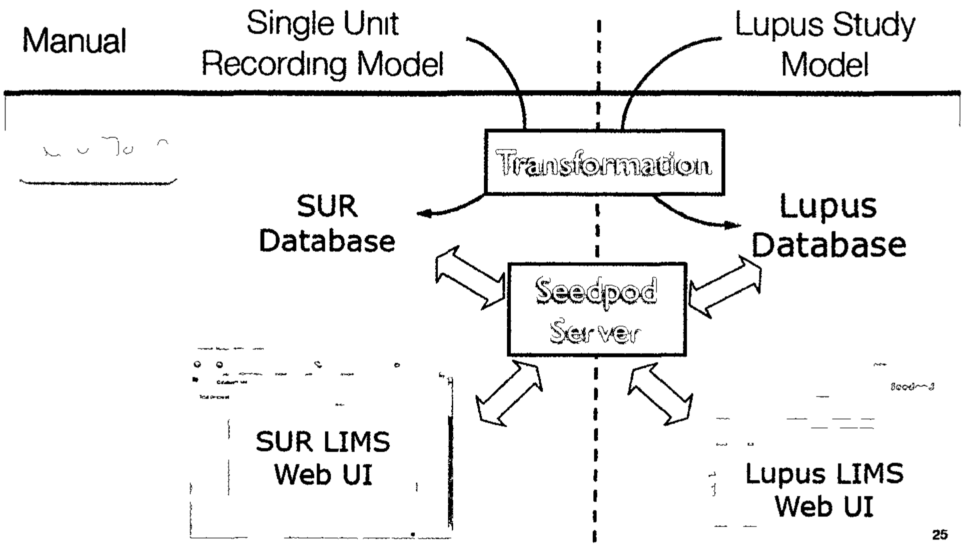


Figure 7.1. The transformation and Seedpod application server are both domain-independent.

4) *Cost saving:* This model-driven approach to building LIMS saves time, development effort, and ultimately cost for research scientists. Required expertise to getting a LIMS running is less. It allows scientists to quickly create a system and start collecting data in a database without worrying about how the data will be used. Informatics teams can better support multiple research labs in an institution level.

7.2. Future Work

The following areas should be undertaken as future work.

1) *System evolution:* System evolution in a MDA LIMS is not well studied. The MDA approach makes the problem simpler by extracting changes to a system into

changes in a model. Future work is needed to test and compare the three approaches described in 6.2.5.

2) *Workflow*: Users cannot model workflow in Seedpod. Teranode (see 3.4.1) integrates the data model with workflow model into one intuitive unit. This idea can be explored and incorporated into Seedpod, creating an IDE mentioned in 6.2.3 that incorporates workflow modeling with data modeling. This IDE would then run on top of Protégé's meta-model and replace the current Protégé GUI.

3) *Query*: Seedpod focuses on data entry as opposed to data analysis. However, it should provide simple basic query functionalities. An interesting problem would be to allow users to phrase their queries through the model and then translate that query to real database query.

4) *Data exporter/importer:* For the purpose of sharing data with collaborators or analysis data using tools with specific data standards, Seedpod should develop a plug-in framework for developers to export and import data sets. For example, a program developer could write an exporter that writes SUR's brain MRI data and meta-data into other MRI data standards for visualization.

5) *Integrating Seedpod LIMS model with knowledge bases:* Experiment LIMS model can be integrated with experiment protocols that are also captured in knowledge models. The LIMS model can use other scientific knowledge bases for references or controlled vocabulary data input. Alternatively, data generated through the experiments may serve as evidence to other knowledge bases. The LIMS model can be re-used in ways that will need to be explored.

Seedpod is a first prototype LIMS building system that incorporates novel techniques for knowledge-model-driven LIMS construction. It is hoped that Seedpod will lead the way for future production systems.

BIBLIOGRAPHY

ADInstruments. (n.d.). *Data Acquisition Software Systems - ADInstruments.* Retrieved 2006, from ADInstruments: ADInstruments

Ambler, S. (2000, July). *Mapping Objects to Relational Databases.* Retrieved 2010, from Agile Data: http://www.agiledata.org/essays/mappingObjects.html

Anderson, N., Lee, S., Brockenbrough, S., Minie, M., Fuller, S., Brinkley, J., et al. (2007, July-Aug). Issues in Biomedical Research Data Management and Analysis: Needs and Barriers. *J Am Med Inform Assoc* , pp. 478–488.

Arnstein, L., Grimm, R., Hung, C.-Y., Kang, J. H., LaMarca, A., Look, G., et al. (2002). System Support for Ubiquitous Computing: A Case Study of Two Implementations of Labscape. *2002 International Conference on Pervasive Computing.* Zurich.

Arnstein, L., Hung, C.-Y., Franza, R., & Zhou, Q. H. (2002). Labscape: A Smart Environment for the Cell Biology Laboratory. *IEEE* , pp. 13-21.

Atzeni, P., Cappellari, P., & Bernstein, P. A. (2005). ModelGen: Model Independent Schema Translation. *ICDE '05 Proceedings of the 21st International Conference on Data Engineering.* Washington DC: IEEE Computer Society .

Bernstein, P. A. (2003). Applying Model Management to Classical Meta Data Problems. *Proceedings of the 2003 CIDR Conference.*

Brinkley, J. (2005). *UW Integrated Brain Project Cortical Stimulation Mapping Database.* Retrieved from Brain Map: http://bmap.biostr.washington.edu/

Brown, A. W. (2004). Model driven architecture: Principles and practice. *Software System Model* , pp. 314-327.

Cho, H., Corina, D., Brinkley, J., Ojemann, G., & Shapiro, L. (2005). A New Template Matching Method using Variance Estimation for Spike Sorting. *2nd International IEEE EMBS Conference on Neural Engineering,* (pp. 225 - 228).

Cohen, J., Dolan, B., Dunlap, M., Hellerstein, J. M., & Welton, C. (2009). MAD Skills: New Analysis Practices for Big Data. *Proc. VLDB Endow,* (pp. 1481-1492).

Dominguez, E., Lloret, J., & Rubio, A. L. (2002). *An MDA-Based Approach to Managing Database Evolution.*

Drexler, E. (2008, 10 25). *The Data Explosion and the Scientific Method.* Retrieved 2010, from Metamodern: http://metamodern.com/2008/10/25/the-data-explosion-and-the-scientific-method/

Entity-attribute-value model. (2010). Retrieved 2010, from Wikipedia: http://en.wikipedia.org/wiki/Entity-attribute-value_model

Estrella, F., Kovacs, A., Goff, J.-M. L., McClatchey, R., & Toth, N. (2001). Meta-Data Objects as the Basis for System Evolution. *CMS Conference Report*, (pp. 1-7).

Estrella, F., Kovacs, Z., Goff, J.-M. L., & McClatchey, R. (2001). Model and Information Abstraction for Description-Driven Systems. *Computing in High Energy and Nuclear Physics.* Beijing.

Fogh, R. H., Bouche, W., Vranken, W. F., Pajon, A., T. J., Bhat, T. N., et al. (2005). A framework for scientific data modeling and automated software development. *Bioinformatics* , pp. 1678–1684.

Fong, C., & Brinkley, J. (2006). Customizable Electronic Laboratory Online (CELO): A Web-based Data Management System Builder for Biomedical Research Laboratories. *AMIA Conference Proceedings,* (p. 922). Seattle.

Free Software Foundation. (2007). *The GNU General Public License 3.0 - GNU Project- Free Software Foundation.* Retrieved 2010, from GNU: http://www.gnu.org/licenses/gpl.html

Gardner, D., & Shepherd, G. M. (2004). A Gateway to the Future of Neuroinformatics. *Neuroinformatics*, pp. 271-4.

Gennari, J. H., Mork, P., & Li, H. (2005). Knowledge Transformations between Frame Systems and RDB Systems. *3rd International Conference on Knowledge Capture (K-CAP'05),* (pp. 197-198). Banff, Alberta, Canada.

Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Crubzy, M., Eriksson, H., et al. (2003, January). The evolution of Pro¬tégé: an environment for knowledge-based system development. *International Journal of Human-Computer Studies* , pp. 89-123.

Gitzel, R., & Korthaus, A. (2004). The Role of Metamodeling in Model-Driven Development. *Proceedings of the 8th World Multi-Conference on Systemics, Cybernetics and Informatics.* Orlando.

Goodman, N., Rozen, S., Stein, L., & Smith, A. (1998). The LabBase System for data management in large scale biology research laboratories. *Bioinformatics* , pp. 562-574.

GraphLogic. (2009). *GraphLogic, Inc.* Retrieved 2010, from GraphLogic: http://www.graphlogic.com/

Gray, J., Liu, D. T., Nieto-Santisteban, M., Szalay, A. S., DeWitt, D., & Heber, G. (2005). *Scientific Data Management in the Coming Decade.* Redmond: Microsoft Research.

Hachem, N., Gennert, M., & Ward, M. (1993). Distributed Database Management for Scientific Data Analysis. *Int. Workshop on Global GIS.*

Hick, J.-M., & Hainaut, J.-L. (2003). Strategy for Database Application Evolution: the DB-MAIN Approach. *Lecture Notes in Computer Science*, pp. 291-306.

Hull, R., & King, R. (1987). Semantic database modeling: survey, applications, and research issues. *ACM Computing Surveys*, pp. 201-260.

I-min A. Chen, V. M. (1995). An Overview Of The Object Protocol Model (opm) And The Opm Data Management Tools. *Information Systems*, pp. 393--418.

Ipad. (2010). *Ipad ELN - Electronic Lab Notebook* . Retrieved 2010, from Ipad ELN: http://www.ipadeln.com/

Jakobovits, R. M., Rosse, C., & Brinkley, J. F. (2002). WIRM: an open source toolkit for building biomedical web appli-cations. *9*(6), 557-70.

Jakobovits, R., Soderland, S. G., Taira, R., & Brinkley, J. (2000). Requirements of a Web-Based Experiment Management System. *Proceedings of AMIA Symposium 2000*, (pp. 374-8).

Kell, D., & Oliver, S. (2004). Here is the evidence, now what is the hypothesis? The complementary roles of inductive and hypothesis-driven science in the post-genomic era. *Bioessays*, 99-105.

Kleppe, A. G., Warmer, J. B., & Bast, W. (2003). *MDA Explained: The Model Driven Architecture(TM): Practice and Promise.* Addison-Wesley.

Kotter, R. (2001). Neuroscience Databases: Tools for Exploring Brain Structure-Function Relationships. *Philosophical transactions of the Royal Society of London*, 1111-20.

LabCentrix. (2007). *LabCentrix - LIMS Consulting Services & Technology Solutions.* Retrieved 2010, from LabCentrix: http://www.labcentrix.com/

Lacroix, Z., & Critchlow, T. (2003). *Bioinformatics: Managing Scientific Data.* Morgan Kaufmann.

Larson, E. (2008, 12 5). *Data-driven Science in the Age of Exponential Information Growth.* Retrieved 2010, from NowPublic: http://www.nowpublic.com/tech-biz/data-driven-science-age-exponential-information-growth

Lazar, J. (2000). *User-Centered Web Development.* Jones & Bartlett Learning.

Liu, C., Orlowska, M., & Li, H. (1997). Realizing Object-Relational Databases by Mixing Tables with Objects. *International Conference on Object Oriented Information Systems,* (pp. 335-346). Brisbane, Australia.

Ludäscher, B., Gupta, A., & Martone, M. (2003). A Model-Based Mediator System for Scientific Data Management. In Z. Lacroix, & T. Critchlow, *Bioinformatics: Managing Scientific Data* (pp. 335--370). Morgan Kaufmann.

ManyDesigns. (2010). *Home of ManyDesigns Portofino.* Retrieved 2010, from ManyDesigns: http://www.manydesigns.com/Home.html

MDA. (2010). Retrieved 2010, from Object Management Group: http://www.omg.org/mda/

Minsky, M. (1974). *A framework for Representation Langauge.* Retrieved 2010, from http://web.media.mit.edu/~minsky/papers/Frames/frames.html

Musen, M. A. (1998, November). Domain Ontologies in Software Engineering: use of protege with the EON architecture. *Methods of Information in Medicine* , pp. 540-550.

Nadkarni, P., Marenco, L., Chen, R., Skoufos, E., Shepherd, G., & Miller, P. (1999). Organization of Heterogeneous Scientific Data Using the EAV/CR Representation. *JAMIA* , pp. 478-493.

Niyomthum, K., & Chittayasothorn, S. (2003). A Transformation from An Object Database to an Object Relational Database. *Proceedings IEEE SoutheastCon* (pp. 7-11). IEEE.

Noah, S. A., & Lloyd-Williams, M. (1995, December). A selective review of knowledge-based approaches to database design. *Information Research* .

Noy, N., & McGuinness, D. (2001). *Ontology Development 101: A Guide to Creating Your First Ontology.* Stanford : Citeseer.

Noy, N., Sintek, M., Decker, S., Crubezy, M., Fergerson, R., & Musen, M. (2001). Creating Semantic Web Contents with Protege-2000. *IEEE Intelligent Systems* , pp. 60-71.

Ojemann, G., Schoenfield-McNeill, J., & Corina, D. (2002, January). Anatomic subdivisions in human temporal cortical neuronal activity related to recent verbal memory. *Nature Neuroscience* , pp. 64-71.

Paszko, C., & Turner, E. (2002.). *Laboratory Information Management Systems.* New York: Marcel Dekker.

Pittendrigh, S., & Jacobs, G. (2001). NeuroSys, A Semistructured Laboratory Database. *Neuroinformatics Journal* , pp. 167-178.

Protege Developer Documentation. (n.d.). Retrieved 11 11, 2009, from http://protege.stanford.edu/doc/pdk/plugins/project_plugin.html

Psychology Software Tools, Inc. (n.d.). *Psychology Software Tools: E-Prime application suite for psychology experiment design, implementation, and analysis.* Retrieved 2006, from Psychology Software Tools, Inc: http://www.pstnet.com/products/e-prime/

Ramakrishnan, R., & Gehrke, J. (2002). *Database Management Systems.* McGraw-Hill .

Rubin, D., Shafa, F., Oliver, D., Hewett, M., & Altman, R. (2002). Representing Genetic sequence data for pharmacogenomics: an evolutionary approach using ontological and relatioinal models. *Bioinformatics*, pp. S207-S215.

Schmidt, D. C. (2006, February). Model-Driven Engineering. *IEEE*, pp. 25-31.

SQL:1999. (2011, January 12). Retrieved 2011, from Wikipedia: http://en.wikipedia.org/wiki/SQL:1999

Stanford Center for Biomedical Informatics Research. (2010). *What is Protégé-2000?* Retrieved 2010, from Protégé: http://protege.stanford.edu/doc/users_guide/

Structural Informatics Group. (n.d.). Retrieved 2010, from Structural Informatics Group: http://sig.biostr.washington.edu/

Swenson, M. (2005 , 5). *Experiment Design Automation: A Potential Solution for Fragmented Informatics in Biopharmaceutical Research and Development.* Retrieved 2006, from Bioscienceworld : http://www.bioscienceworld.ca/ExperimentalDesignAutomation

Tang, Z., Kadiyska, Y., Li, H., Suciu, D., & Brinkley, J. F. (2003). Dynamic XML-Based Exchange of Relational Data: Application to the Human Brain Project. *AMIA Annu Symp Proc,* (pp. 649–653).

Teranode. (2010). *Teranode Incorporated.* Retrieved 2010, from Teranode: http://teranode.com/

The Apache Software Foundation. (2011). *The Apache Web Server Project.* Retrieved 2011, from Apache: http://httpd.apache.org/

Wikipedia: Ajax. (n.d.). Retrieved 2010, from http://en.wikipedia.org/wiki/Ajax_(programming)

Wikipedia: Java Platform Enterprise Edition. (n.d.). Retrieved 2010, from http://en.wikipedia.org/wiki/Java_Platform,_Enterprise_Edition

Wikipedia: Web application. (2009, September 3). Retrieved September 3, 2009, from http://en.wikipedia.org/wiki/Web_application

XMIBackendTechnicalBackground. (2006). Retrieved 2010, from Protege Wiki Page: http://protege.cim3.net/cgi-bin/wiki.pl?XMIBackendTechnicalBackground

CURRICULUM VITAE

Hao Li

EDUCATION

University of Washington

- PhD in Biomedical and Health Informatics (2011).
    - Awarded National Library of Medicine Training Grant (June 2004 – June 2007).
- BS in Biochemistry and Neurobiology. (2001).

TECHNICAL AND RESEARCH EXPERIENCES

MITRE Corporation McLean, VA

Senior Database Technology Software Engineer January 2008 - present

- Data integration engineer
    - Develop system specifications for distributed database systems.
    - Develop common vocabulary for large military databases.
    - Develop workflow process for data integration.
- Database and software development in OpenII[2]

[2] OpenII is an open source data integration framework and application developed at MITRE in collaboration with Google and other academic institutes. Harmony and Unity are modules of OpenII. http://openii.sourceforge.net/

  - Develop schema importers and exporters for OpenII
  - Develop a database and API for multiple flight sensor requesting messages.
  - Key developer of Unity, a semi-automated vocabulary generation application in OpenII.
- A liaison between customers and MITRE research project
  - Customize Harmony to meet the needs of a large military customer.
  - Contribute to OpenII research and development from real user experiences.
- Technology advisor to the National Center Research Resources at National Institute of Health
  - Investigate current research and development in Biomedical Informatics.
  - Publish technology review papers.
- Technology advisor to the Neuro Names knowledge base design at George Mason University

Structural Informatics Group, University of Washington Seattle, WA

Research Assistant September 2001 – December 2007

- Project Seedpod (PhD dissertation project)
  - Design and development of a general purpose web-based solution that helps scientists manage data in a relational database by simply modeling the lab data without the need of computer programming. Model using Protégé-

2000, model-driven web-based application implemented in JAVA and PostgreSQL.

- XBrain Project (CS graduate level database course project)
  - Developed database infrastructure for publishing human brain data stored in relational databases dynamically in XML using SilkRoute.

Teranode Corporation Seattle, WA

Consulting Engineer Internship June 2005 - December 2005

- Developed an integrated and automated Teranode platform solution to manage samples and workflow at University of Washington Center of Expression Array. Duties included gathering user requirements, requirement analysis, data and workflow modeling, and solution customization.

Next Generation Internet Project, University of Washington Seattle, WA

Undergraduate Internship/ Student Programmer June 2000 - December 2001

- Designed and developed a web-based information management application to manage case presentation and images, which enabled the Seattle Cancer Care Alliance weekly multi-location tumor teleconferences. Application implemented using PHP and MySQL database.

PUBLICATIONS

- Rosenthal A, Mork P, Li H, Stanford J, Koester D, Reynolds P. Cloud Computing: A New Business Paradigm for Biomedical Informatics. Not yet in print. Accepted for publication in Journal of Biomedical Informatics, April 2009.
- Smith K, Morse M, Mork P, Li M, Rosenthal A, Allen D, Selligman L. The Role of Schema Matching in Large Enterpriese. CIDR 2009, Monterey, CA.
- Mork P, Stanford J, Li H, Smith K. Sharing Data Containers in Translational Research. Proceedings of AMIA Spring Congress, 2008, Phoenix, AZ.
- Li H, Gennari JH, Brinkley JF. Model Driven Laboratory Information Management Systems. Proceedings of the AMIA Conference, 2006, Washington, DC.
- Gennari, J, Mork, P, and Li H Knowledge Transformations between Frame Systems and RDB Systems. Proceedings of the K-CAP05 Conference, 2005.
- Li H, Brinkley JF, and Gennari J. Semi-automatic Database Design for Neuroscience Experiment Management Systems. Proceedings of MedInfo 2004, San Francisco, CA.
- Tang Z, Kadiyska Y, Li H and Suciu, D and Brinkley, JF (2003) Dynamic XML Based Exchange of Relational Data: Application to the Human Brain Project. In *Proceedings, American Medical Informatics Association Fall Symposium.* In press.
- Li H, Lober WB, et al. Iterative Development of a Web Application to Support Teleconferencing of a Distributed Tumor Board. In Proceedings, AMIA 2002, San Antonio, TX.
- Lober WB, Li H, Trigg LJ, Stewart BK, Chou D. Web Tools for Distributed Clinical Case Conferencing. Proceedings of AMIA Annual Symposium, 959, 2001.

SCHOLARLY ACTIVITIES

- Reviewer for Journal of Biomedical Informatics (2010)
- Program Committee member of Data & Knowledge Engineering Journal Elsevier special issue on Contribution of Ontologies in Designing Advanced Information Systems (2009)

HOBBIES AND INTERESTS

- Travel, photography, rowing, bicycling, hiking, cooking, art, coffee.
- Chinese dance and ballet. A dance performer at Asian Art Performing Center (1994-2006)